XILINX®

ONLINE

XEPLD REFERENCE GUIDE

● TABLE OF CONTENTS

● INDEX

GO TO OTHER BOOKS

XACT™ STEP

0401416

Contents

Chapter 1 XEPLD Functional Description

Product Description.......... 1-1
The XACT Design Manager (XDM) 1-2
Design Methodology 1-5
    Designing with Schematic Capture Tools 1-7
    Designing with Behavioral Entry Tools 1-9
    Converting Existing PAL Designs 1-10

Chapter 2 XACT Design Manager Menus

Using XDM.......... 2-1
    The Program List File 2-1
    Using XDM on a PC.......... 2-2
    Using XDM on a Workstation.......... 2-4
        About X-Windows and Graphic Interfaces.......... 2-4
        Mouse Configuration 2-5
        Window Operations 2-5
        Window Buttons.......... 2-5
        Window Shortcuts.......... 2-6
        Active Window 2-6
        Starting XDM 2-6
    Entering Commands 2-8
        The Graphic Interface.......... 2-9
        The Command Line Interface 2-9
Task-Based Command Flows.......... 2-10
    Schematic Design to Programming File 2-10
    About the XEMAKE Command.......... 2-11
    Behavioral Design to Programming File 2-12
    JEDEC PAL File to PLUSASM PAL File.......... 2-13
    Workview Simulation 2-13
        Functional Simulation 2-13

Timing Simulation 2-14
Timing Simulation for a Behavioral Design 2-14
OrCAD Simulation 2-14
Functional Simulation 2-14
Timing Simulation 2-15
Timing Simulation for a Behavioral Design 2-16
About the XSIMMAKE Command 2-16
Menu Tour 2-16
The Settings Fields 2-17
The Design Entry Menu 2-18
The Translate Menu 2-18
XEMAKE (EPLD Automatic Implementation Tool) 2-18
ABL2PLD (Create EPLD from Xilinx ABEL Design) 2-20
ANNOTATE (Annotate SCH — PC Only) 2-20
CLEANUP (Cleanup SCH — PC Only) 2-20
INET (Compile SCH — PC Only) 2-20
SDT2XNF (Create Orcad Netlist — PC Only) 2-21
WIR2XNF (Create Workview Netlist) 2-21
JED2PLD (Import & Assemble PLD JEDEC) 2-22
PLUSASM (Assemble PLD Equations) 2-22
PINSAVE (Save EPLD Pinout) 2-23
XNFMERGE (Merge Multiple XNF Files) 2-23
The Fitter Menu 2-24
FITEQN (Integrate EPLD Behavioral Design) 2-24
FITNET (Integrate EPLD Schematic Design) 2-25
PALCONVT (Convert PAL-based Design) 2-26
The Verify Menu 2-28
XSIMMAKE (Create Simulation File) 2-28
ASCTOVST (OrCAD/VST Utility — PC Only) 2-29
ORCAD (OrCAD VST — PC Only) 2-29
XNF2VST (Create Orcad Sim Files — PC Only) 2-29
VSM (Viewsim Wirelister) 2-30
XNF2WIR (Create Viewlogic Sim Files) 2-31
MAKEJED (Make JEDEC Programming Files) 2-31
MAKEPRG (Make Hex Programming Files) 2-32
VMH2XNF (Make XNF for Timing Simulation) 2-32
PROLINK (Program EPLD Device — PC Only) 2-33
The Utilities Menu 2-33
Browse 2-34
DirClean 2-34
Directory 2-34

DOS (PC only) ... 2-35
Edit ... 2-35
Execute ... 2-36
Help ... 2-36
Report ... 2-37
ScanDisk ... 2-37
Version ... 2-37
The Profile Menu ... 2-38
Cursor ... 2-38
Family ... 2-38
KeyCursor ... 2-38
Keydef ... 2-38
Menucolors ... 2-39
Mouse ... 2-39
Options ... 2-39
Palette ... 2-40
Part ... 2-40
Printer ... 2-40
Readprofile ... 2-40
Saveprofile ... 2-40
Settings ... 2-40
Speed ... 2-40

Chapter 3 Report Formats

Viewing Reports ... 3-1
The Resource Report ... 3-2
Logic Resources ... 3-2
Required Pin Resources ... 3-2
Used Pin Resources ... 3-2
Remaining Pin Resources ... 3-2
Fast Inputs and Outputs ... 3-2
The Mapping Report ... 3-4
Function Name ... 3-4
Macrocell Location ... 3-4
Pkg Pin ... 3-4
Pin Use ... 3-5
Power Estimation (7300 Family Only) ... 3-5
The Pinlist Report ... 3-7
Pkg Pin ... 3-7
Pin Type ... 3-7
Pin Use ... 3-7

Pin Name 3-7
Pin Use Legend 3-7
The Partitioner Report 3-9
Summary 3-9
Part Name 3-9
Number of Outputs 3-9
Number of Input Lines Used 3-9
Signal Inputs (Complete Design Only) 3-10
Number of Shared Pt 3-10
O/IO Used (Complete Design Only) 3-10
O/IO Avail (Complete Design Only) 3-10
Size Factor 3-10
Inputs Used by Each Partition 3-10
Partition Listing 3-10
Signals Used 3-10
Anded UIM Inputs Used (Complete Design Only) 3-11
Inputs Used by Each Output Table 3-11
MC No 3-11
Output Name 3-11
Pin Req (Complete Design Only) 3-11
Pin Avl (Complete Design Only) 3-11
Sh Pt 3-11
Input Listing 3-11
The Logic Optimizer Report 3-16
Summary 3-16
Device Specific Optimization 3-17
Outputs that Were Collapsed 3-17
Outputs Removed from the Network 3-17
The PLUSASM Assembler Log Report 3-22
Product Term Allocation (PLFB9/PLFFB9 Format) 3-22
Pin 3-22
Name 3-22
Type 3-23
Local P-terms Available 3-23
Local P-terms Used 3-23
Shared D1 P-terms Used and Shared D2 P-terms Used .. 3-23
Total P-terms Used 3-23
Partitioner Log Report (Standard PAL Format) 3-23
The PAL Interconnect Report 3-29
PAL Pin 3-29
Signal Name 3-29

PAL Use 3-29
Chip Use 3-29
Connectivity 3-29
Unconnected Pins 3-30
Summary 3-30
The EQN File 3-33

Chapter 4 PLUSASM Command Reference

Introduction 4-1
PLUSASM Overview 4-1
PLUSASM File Structure 4-2
The Header Section 4-4
Header Statements 4-4
The Declarations Section 4-5
CEPIN 4-6
CHIP 4-7
file_name 4-7
device_name 4-7
signal_list 4-8
FASTCLOCK 4-10
FASTINPUT 4-12
FOEPIN 4-13
INCLUDE_EQN 4-15
INPUTPIN 4-16
RCLK = fastclock_name 4-16
CE = ce_name 4-16
LE = fastclock_name 4-16
FI 4-16
IOPIN 4-18
RCLK = fastclock_name 4-18
CE = ce_name 4-18
LE = fastclock_name 4-18
PINFBK 4-18
FOE = foe_name 4-19
NODETRST 4-19
LOGIC_OPT 4-21
MINIMIZE 4-23
MRINPUT 4-24
NODE 4-25
OPTIONS 4-27

OUTPUTPIN 4-30
FOE = foe_name 4-30
NODETRST 4-30
PARTITION 4-32
Physical PARTITION Statements 4-32
Logical PARTITION Statements 4-33
Linked Equations 4-34
General Rules for PARTITION Statements 4-35
Logical Partitions Using Less Than 9 Macrocells 4-36
Logical Partitions Using More Than 9 Macrocells 4-36
Physical Partitions Using Less Than 9 Macrocells 4-36
Physical Partitions Specifying Starting Macrocells 4-37
PWR 4-38
STRING 4-39
The Equation Section 4-40
Combinatorial and Registered Equations 4-40
Control Equations 4-43
.ADD (Arithmetic Carry Enable) 4-44
.ADDMODE (Arithmetic Carry Enable) 4-46
.CLKF (Register Clock Source) 4-48
.D1 and .D2 (ALU Inputs) 4-50
Using the ALU 4-50
.EXPORT (FFB Product Term Assignment) 4-52
.FBK (Local Feedback) 4-54
.FBKINVERT (Invert Macrocell Feedback) 4-55
.FI (FastInput Source) 4-56
.PIN (PIN Input Source) 4-57
.PRLD (Preload State) 4-58
.RSTF (Asynchronous Reset) 4-59
.SETF (Asynchronous Set) 4-60
.SHIFT (Local Shift) 4-61
.T (Toggle Flip-Flop Specification) 4-64
.TRST (3-State Control) 4-65
Defining Signal Polarity in Equations 4-66
Output Signal Polarity 4-66
Input Signal Polarity 4-67
PLUSASM Syntax 4-68
Notation 4-68
Delimiters 4-68
Operators and Special Characters 4-68
Comments 4-69

Reserved Characters 4-69
Names 4-69
Reserved Words 4-70
Unsupported PALASM Syntax 4-70
Parenthesis 4-70
State Machine Syntax 4-71
The Latched Output Equation Operator 4-71
Simulation Control Statements 4-71
Device Specific Syntax 4-71
PLUSASM Command Syntax Quick Reference 4-72
Declaration Statements Used in Schematic PLDs and Include Files 4-73
Declaration Statements Used in Top-Level Files 4-74
Equation Statements Used in High Density Function Blocks . 4-75
Equation Statements Used in Fast Function Blocks 4-76

Chapter 5 XEPLD Fitter Modules and Files

XEPLD Fitter Modules 5-1
The Netlist Reader (Schematic Only) 5-3
The Muncher (Schematic Only) 5-3
The Logic Optimizer 5-4
PLUSASM, the Partitioner, and the Minimizer 5-5
The Chip Builder 5-7
The Design Rule Checker 5-7
General Design Rule Violations 5-7
Pad Component Design Rule Violations 5-8
FastCLK, Clock Enable, and Fast Output Enable Violations 5-8
The Gate Optimizer 5-9
The Mapper 5-9
The Interconnector 5-9
XEPLD Files and Directories 5-10
The Netlist File 5-15
The PLUSASM Equation Files 5-15
JEDEC Source Files 5-15
Design Database File 5-15
The Report Files 5-15
The Log Files 5-16
The Error/Warning Files 5-16
The Programming Bit-map File 5-16

Appendix A Typical Component Equations

Basic Gates A-1
4-Bit Counter (CB4X2) A-2
4-Bit Accumulator (ACC4X1) A-3
4-Bit Adder (ADD4X1) A-4
Flip-Flop (FDSRE) A-5
Latch (LD) A-6
Multiplexer (M4_1E) A-7
Comparator (COMP4) A-8
Magnitude Comparator (COMPM4) A-9
Shifter (SR4RLED) A-10
T Flip-Flop (FTC) A-11
Decoder A-12

Appendix B PAL Devices Supported

Specific and Generic PAL Symbols for Schematics B-1
PAL Equation File Syntax B-2
22V10 and 20V8 Support B-3
  Default Clock B-3
  Default 20V8 3-State Control B-3
  Global 22V10 Set/Reset B-3
  Automatic Inversion of Set and Reset B-4
Generic PAL Support B-4
PLFB9 and PLFFB9 Support B-5
  Clocks B-5
  PLFB9 Arithmetic Carry-In and Carry-Out Pins B-5
  Defining PLFFB9 Fast Inputs B-5
PALs Supported through Generic PAL Components B-6

Appendix C Equation Entry Application Note

Introduction C-1
  How to Follow this Tutorial C-1
  The Tutorial Design C-2
  Overview of the Sessions C-3
  The Example Files C-4
Session 1: Using the XEPLD Software C-4
  Step 1: Prepare the System C-4
    Setting Up the PC C-5
    Setting Up the Workstation C-5
  Step 2: Start XDM C-6

Step 3: Select Menu Items C-7
Step 4: Configure the XEPLD Environment C-9
Configuring the PC C-9
Configuring the Workstation C-10
Session 2: Design Entry C-11
Step 1: Segment the Design C-11
Step 2: Create Boolean Equations C-12
Step 3: Consolidate the Design C-16
Step 4: Assign Signals to Specific EPLD Pins C-19
Session 3: Fitting the Design C-20
Step 1: Invoke the Fitter C-20
Step 2: View the Reports C-21
Step 3: Save Pin Assignments C-22
Step 4: Create the Programming File C-22
Step 5: Create a Simulation Model C-23

Appendix D Glossary D-1

Index Index-1

Trademark Information

XEPLD Functional Description

Product Description

The XEPLD Design System from Xilinx allows you to create designs for Xilinx EPLD devices. No other tools are required. However, you can also use third-party schematic capture tools and/or PLD compilers to enter and verify your design.

You can use schematics, behavioral descriptions, or a mixture of both to represent your design. XEPLD software translates and maps the design quickly and automatically onto the Xilinx EPLD device you choose, creates a programming file for the device programmer, and provides reports and simulation files for design analysis and verification.

To support schematic entry, Xilinx provides a component symbol library for each of the supported schematic capture tools. In addition to TTL-like components, this library contains industry-standard PAL symbols (for example, 22V10 and 20V8) to simplify direct PAL-based design conversions. XEPLD software produces models of the completed design for each of the supported simulators.

You can enter designs behaviorally using third-party PLD compilers, then export the designs to XEPLD in the form of equation files. Alternatively, you can enter equations directly using a text editor.

The rest of this chapter describes the parts of the XEPLD Design System and how they work together. It is divided into four sections:

- The XACT Design Manager (XDM) — A brief description of the XEPLD software's user interface.
- Design Methodology — The basic steps for creating a schematic or behavioral design and an explanation of how XEPLD software supports PAL conversion.
- XEPLD Fitter Modules — A description of each of the modules within the XEPLD software.
- XEPLD Files and Directories — A description of the files and directories that comprise the XEPLD software.

The XACT Design Manager (XDM)

The XACT Design Manager (XDM) is a set of menus with commands that you use to program an EPLD. Via the XDM interface, you can access all the functions of the XEPLD software, your CAE tools, text editors, netlist and behavioral translators, and programmer control software. Figure 1-1 shows the relationships between XDM and the various parts of the XEPLD software system.

The XDM main menu (Figure 1-2) and submenus display all the operational choices available to you. You can use either the mouse or the command line to select commands. XDM provides a convenient structure for organizing all the files associated with your designs: all the files associated with a particular design have the same name, but different extensions. For more about XDM, see the "XACT Design Manager Menus" chapter of this manual.

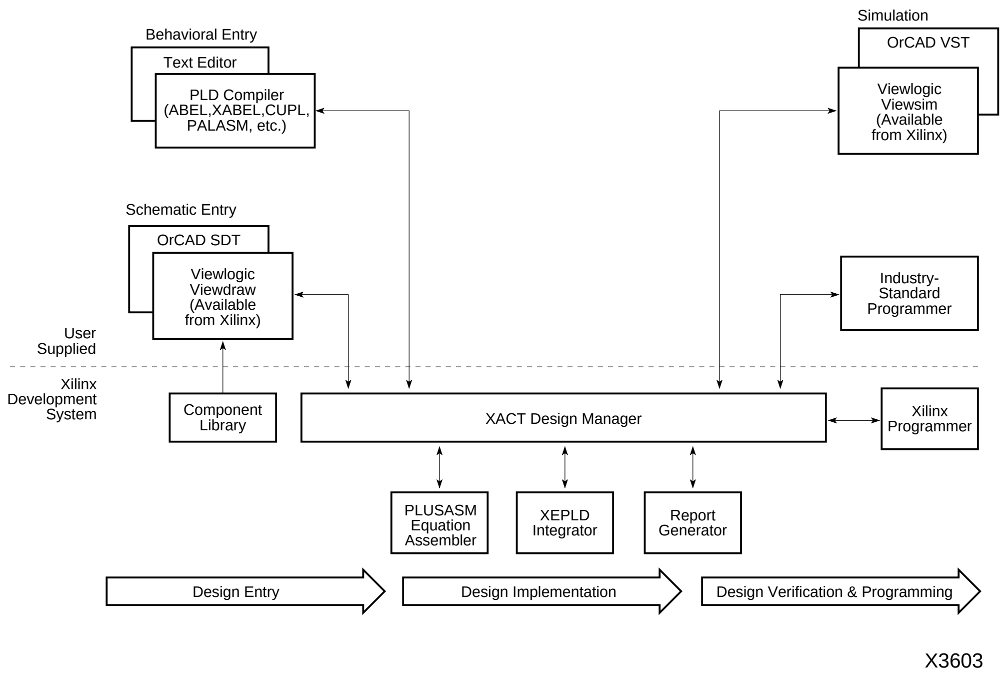
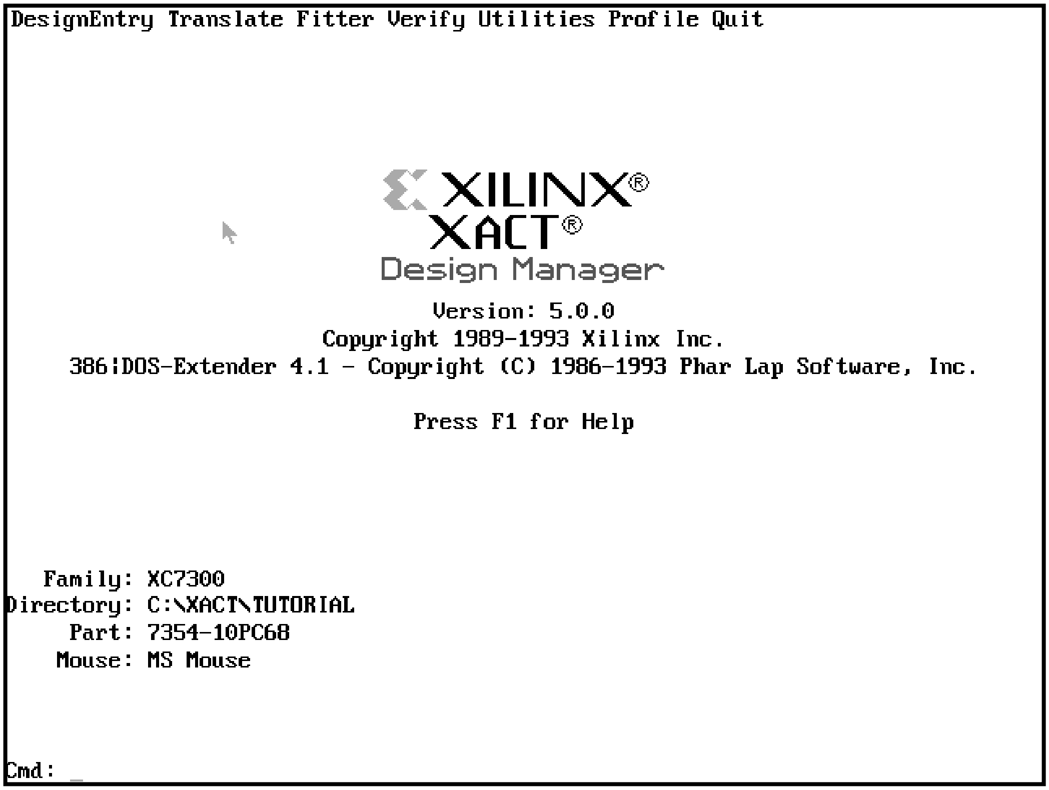

Behavioral Entry
Text Editor
PLD Compiler (ABEL,XABEL,CUPL, PALASM, etc.)
Simulation
OrCAD VST
Viewlogic Viewsim (Available from Xilinx)
Schematic Entry
OrCAD SDT
Viewlogic Viewdraw (Available from Xilinx)
Industry-Standard Programmer
User Supplied
Xilinx Development System
Component Library
XACT Design Manager
Xilinx Programmer
PLUSASM Equation Assembler
XEPLD Integrator
Report Generator
Design Entry
Design Implementation
Design Verification & Programming
X3603

Figure 1-1 XDM Interfaces with Third-Party Tools



```
DesignEntry Translate Fitter Verify Utilities Profile Quit
```

```
                          Version: 5.0.0
                  Copyright 1989-1993 Xilinx Inc.
    386|DOS-Extender 4.1 - Copyright (C) 1986-1993 Phar Lap Software, Inc.

                         Press F1 for Help



   Family: XC7300
Directory: C:\XACT\TUTORIAL
     Part: 7354-10PC68
    Mouse: MS Mouse


Cmd: _
```

Figure 1-2   The XACT Design Manager Menu

Some third-party design tools have their own design manager, as shown in the following figure:

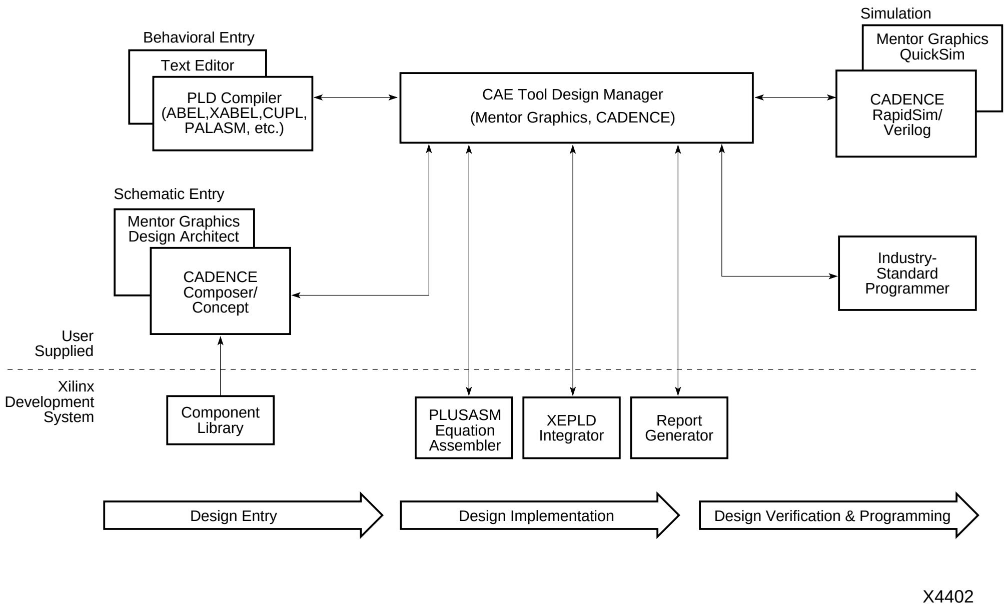

Behavioral Entry
Text Editor
PLD Compiler (ABEL,XABEL,CUPL, PALASM, etc.)
CAE Tool Design Manager
(Mentor Graphics, CADENCE)
Simulation
Mentor Graphics QuickSim
CADENCE RapidSim/ Verilog
Schematic Entry
Mentor Graphics Design Architect
CADENCE Composer/ Concept
Industry-Standard Programmer
User Supplied
Xilinx Development System
Component Library
PLUSASM Equation Assembler
XEPLD Integrator
Report Generator
Design Entry
Design Implementation
Design Verification & Programming
X4402

Figure 1-3 Third-Party Design Tools that Bypass XDM

Design Methodology

Designing circuits using XEPLD is much like designing with TTL devices or programmable logic devices (PLDs). You can express your designs as schematics (Figure 1-4) or in behavioral form (Figure 1-5).

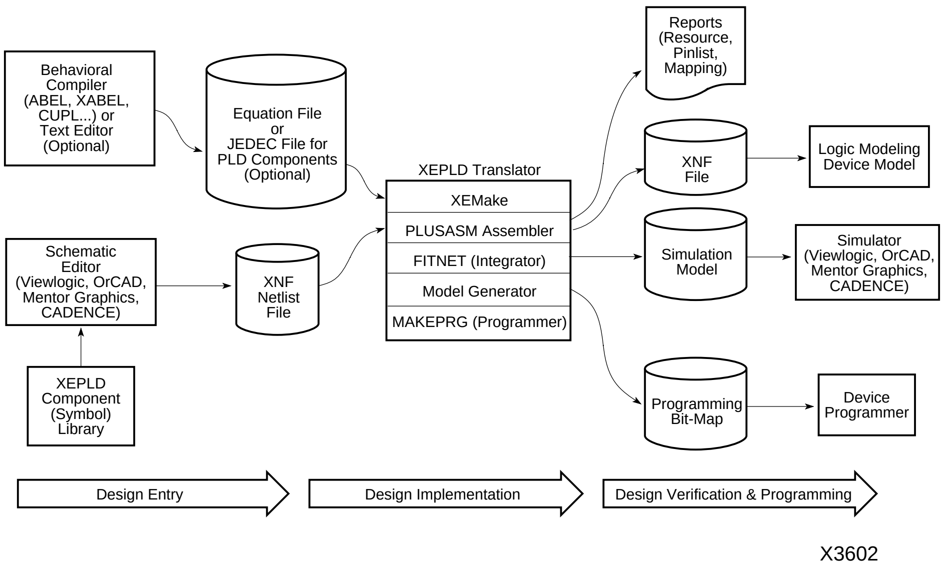

Behavioral Compiler (ABEL, XABEL, CUPL...) or Text Editor (Optional)
Equation File or JEDEC File for PLD Components (Optional)
Schematic Editor (Viewlogic, OrCAD, Mentor Graphics, CADENCE)
XNF Netlist File
XEPLD Component (Symbol) Library
XEPLD Translator
XEMake
PLUSASM Assembler
FITNET (Integrator)
Model Generator
MAKEPRG (Programmer)
Reports (Resource, Pinlist, Mapping)
XNF File
Logic Modeling Device Model
Simulation Model
Simulator (Viewlogic, OrCAD, Mentor Graphics, CADENCE)
Programming Bit-Map
Device Programmer
Design Entry
Design Implementation
Design Verification & Programming
X3602

Figure 1-4 Designing with Schematic Capture Tools

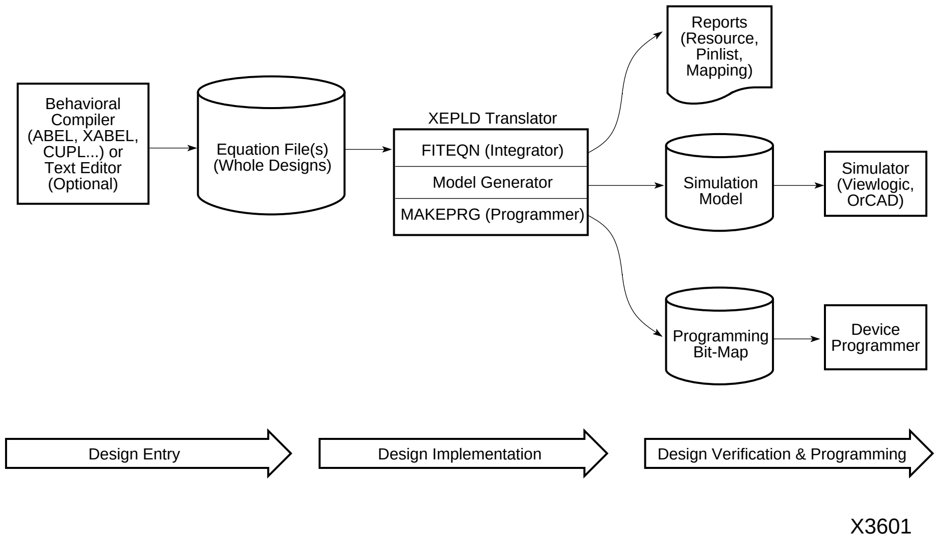

Behavioral Compiler (ABEL, XABEL, CUPL...) or Text Editor (Optional)
Equation File(s) (Whole Designs)
XEPLD Translator
FITEQN (Integrator)
Model Generator
MAKEPRG (Programmer)
Reports (Resource, Pinlist, Mapping)
Simulation Model
Simulator (Viewlogic, OrCAD)
Programming Bit-Map
Device Programmer
Design Entry
Design Implementation
Design Verification & Programming
X3601

Figure 1-5 Designing with Behavioral Entry Tools

Designing with Schematic Capture Tools

To express a design in schematic form, follow this basic procedure:

1. **Select the XC7200 or XC7300 Library.** Make sure you are using the XEPLD component library created for your schematic design entry package. The XC2000, XC3000, and XC4000 libraries are for FPGAs.

   Most of the symbols in the XC7200 and XC7300 libraries are generic; they are present in all Xilinx libraries, allowing easy porting of designs from one device family to another. Some of the symbols are EPLD-specific, allowing you to take advantage of special features of the EPLD architecture.

2. **Select components.** Develop the design by selecting the appropriate components from the XEPLD library. For custom functions that are not available in the library, use PLD library components such as 22V10, or create your own symbol and draw a lower-level schematic under it.

   The XEPLD software supports conventional ASIC gate level design. You can enter a complete design with no boolean equation input.

3. **Enter your design.** Use a Xilinx-supported third-party schematic capture tool to enter your design. You can use attributes to control logic optimization and take advantage of EPLD architectural features.

4. **Specify pin preferences.** You can specify your pin assignment preferences for I/O signals in the schematic. For better fitting, you can let the fitter assign the pins for you.

5. **Define custom logic functions.** You can do so in one of two ways:

   - You can define a lower-level schematic for a custom macro symbol.
   - You can use a commercial PLD compiler or supply PLUSASM Boolean equations. If you use a commercial PLD compiler, you need to export PALASM, PLUSASM, or JEDEC files to XEPLD. You can use a library PLD symbol or create a custom primitive symbol to refer to this equation file.

6. **Perform functional simulation.** You can perform functional simulation only if your design is purely schematic, that is, it has no

PLD equation files. After verifying the functionality of your design, you can return to your schematic files to fix the logic of your design. For more information, see the *Interface User Guide* for your CAE tool.

7. **Export the netlist and fit the design.** The XEPLD software reads the schematic netlist (and the PLUSASM equation files or JEDEC maps if you have included them), translates them, maps your design, and generates a programming map for the target device. The entire processing is performed by the XEPLD Fitter, which is a collection of many software modules each with a specific function. For more information on these modules, refer to the "XEPLD Fitter Modules and Files" chapter.
8. **Examine the reports.** XEPLD generates reports about actual resource utilization and I/O pin mapping. You can return to your schematic or equation files to add or remove resources and refit your design.
9. **Perform timing simulation.** If you are happy with the reports, you can proceed with timing simulation to verify the functionality and timing of your design. After simulation, you can return to your schematic or equation files to fix the logic of your design and rerun the Fitter.
10. **Perform logic modeling.** The XNF file produced in the fitting process can be used with third-party logic modeling software.
11. **Program the device.** When the design is correct and is successfully mapped, you can download the programming map on a supported device programmer and program the target EPLD device.
12. **Freeze the pinout.** After prototyping, you can "freeze" the pinout of your existing design before making some adjustments to your logic design. By freezing the pinout, you may be able to avoid or minimize PC board changes.

For more information on the design entry techniques for schematic-based design, see the *Interface User Guide* for your CAE tool.

Designing with Behavioral Entry Tools

If you do not wish to use schematic capture tools, you can express your design in behavioral form.

1. **Enter your design.** You have the following choices:

   - You can read designs expressed in the PLUSASM-compatible format from third-party behavioral design tools, including PLD compilers (such as ABEL, Xilinx ABEL, CUPL, PALASM, LOG/iC, etc.).
   - You can enter the design directly in the PLUSASM language.

   The PLUSASM equation file syntax is the interface language XEPLD uses for all behavioral design. PLUSASM is a superset of the boolean logic language used by the industry-standard PALASM assembler. PLUSASM provides you with access to the advanced architectural features of the Xilinx EPLD devices. PLUS-ASM also allows you to control the mapping process, including partitioning and minimization.

   Here are some examples of the flexibility you have in expressing your design:

   - You can express a design as either a single, large behavioral description or you can break it down into smaller, manageable blocks the size of PLD-like functions.
   - You can enter your entire design in high-level language or state machine formats using commercial PLD compilers, or directly in PLUSASM equation syntax using a text editor. If you use commercial PLD compilers, you can use their functional simulation capability to test the logic of your design before mapping.
   - You can use PLUSASM language constructs to explicitly control the mapping of any or all of your logic into the device or to specify a preferred pinout.

2. **Fit your design.** The XEPLD software reads equation files and maps the design to the Xilinx EPLD device. The XEPLD software has logic optimization, device optimization, automatic partitioning, and Boolean minimization routines to handle large equation files and to utilize logic resources most efficiently.

3. **Examine the reports.** After the fitting process, you can look at the reports XEPLD generates to find out how the equations were minimized, partitioned, and mapped onto the target architecture. You can also find out the actual resource utilization and I/O pin mapping.
4. **Simulate your design.** If you also have a supported simulator, you can simulate your design after mapping, with full timing accuracy, using a model generated by XEPLD.
5. **Program the device.** When the design is successfully mapped, you can download the programming bitmap file to a supported device programmer and program the target EPLD device.
6. **Freeze the pinout.** After prototyping, you can "freeze" the pinout of your existing design before making some adjustments to your logic design. By freezing the pinout, you may be able to avoid or minimize PC board changes.

For more information on the design entry techniques for behavioral designs, refer to the “PLUSASM Command Reference” chapter of this manual and the *XEPLD Design Guide*.

Converting Existing PAL Designs

The Xilinx EPLD architecture and the XEPLD software allow you to easily incorporate existing multiple PAL-based designs into an EPLD to reduce power, reduce costs, increase reliability, and simplify PC board layout. The Xilinx XC7000-series EPLD devices contain PAL-like function blocks (FBs) that are linked by a 100% routable interconnection matrix. This EPLD architecture, like PALs, has fixed, predictable delays. Each FB can be thought of as a 24V9 component (a 24 input AND-OR array with 9 outputs). This architecture allows you to combine existing PAL-based designs into an EPLD device, with little or no modification, and achieve predictable performance results.

You can use your PLD compiler (such as ABEL) to export each of your PAL designs in a PALASM- or PLUSASM-compatible equation file format. You can then use the PALCONVT command to create a simple PLUSASM file that links all your PALs and defines the EPLD device I/O interface. The XEPLD software can read these files directly.

You can also convert a PAL-based design using schematic entry. The

XEPLD component library contains symbols for standard PALs, such as 20V8 and 22V10, and generic PLD symbols to represent other PALs. You define the logic for these PAL components just as you do for behavioral entry. Your schematic is drawn the way the PAL device would be interconnected on the board.

The XEPLD software automatically combines your original PAL equation files into one comprehensive body of equations that is minimized and partitioned to make the most efficient use of device resources. This allows you to concentrate on your logic design without concern for physical design partitioning or placement in the EPLD device.

For more information and examples of PAL design conversion, see the "Equation Entry Application Note" appendix of this manual and the *XEPLD Design Guide*.

Chapter 2

XACT Design Manager Menus

XDM, the XACT Design Manager, serves as a menu-driven shell for executing all XACT development operations, including the XEPLD Fitter and related interfaces.

The following sections explain the features of XDM:

- *Using XDM* — This section illustrates how to invoke and operate the XDM menu system.
- *Menu Tour* — This section provides a tour to assist you in using the menus.

Using XDM

This section tells you about the file XDM uses to build its menus, how to start up and use XDM, and how to enter commands.

Invoking the XACT Design Manager is different for PC-type systems and workstations. Please read the sections that relate to your system.

When used in a workstation environment, XDM requires that X-Windows, Openlook/Motif, or the HP Design Manager be running.

The Program List File

XDM maintains a program list file (proglist.xdm) to minimize the time required for start up. This file contains a list of XDM-supported programs (executable files) installed on your system.

When generating proglist.xdm, XDM attempts to write it in the following order:

1. To the directory specified by the XACTUSER environment variable.

2. To the “data” subdirectory in the directory specified by the XACT environment variable.
3. To the current working directory.

When reading proglist.xdm, XDM searches for it in the following order:

1. In the current working directory.
2. In the directory specified by the XACTUSER environment variable.
3. In the “data” subdirectory in the directory specified by the XACT environment variable.

Using XDM on a PC

The XDM executable is located in the XACT software directory (default C:\XACT), which should be included in the PATH specified in your autoexec.bat file.

To invoke XDM, enter the following command at your operating system command prompt.

```
XDM
```

Once the XDM command is entered, the Design Manager software loads into memory and reads the proglist.xdm file to set up your menus. While this is being done, a message similar to the following will be displayed.

```
Reading c:\xact\data\proglist.xdm...
```

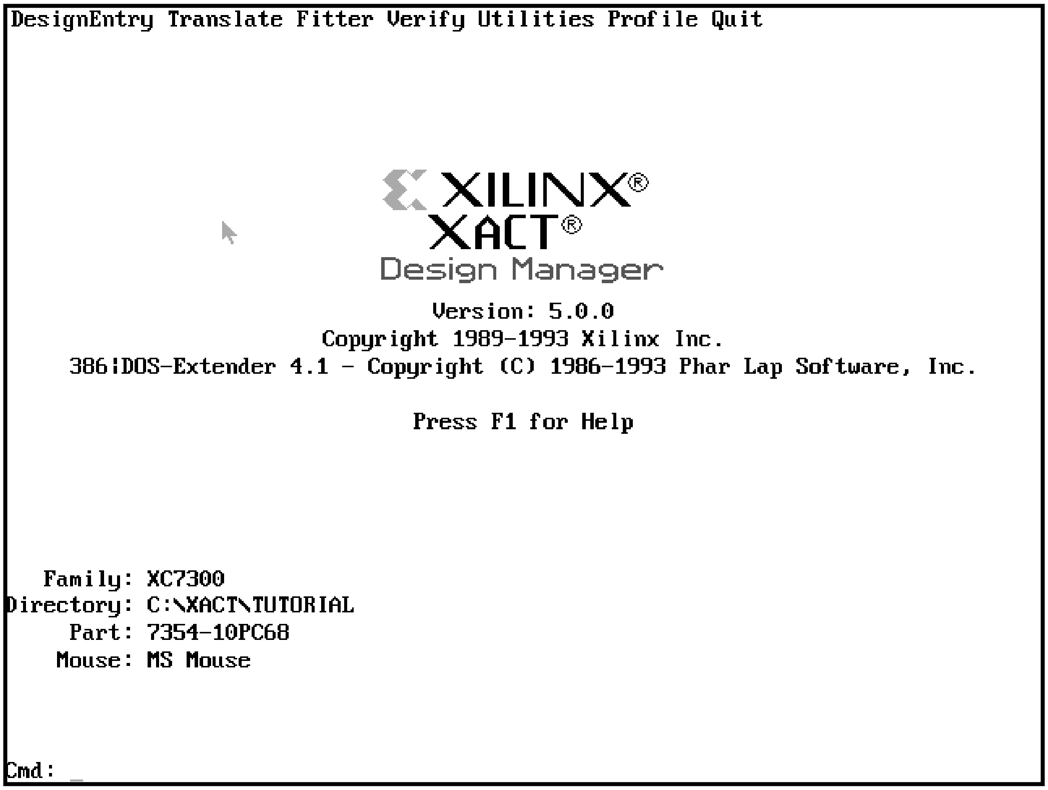
If proglist.xdm is missing, as in the very first invocation of XDM, it will be automatically generated by XDM. See the ScanDisk entry in “The Utilities Menu” in the “Menu Tour” section in this chapter.

After a short while, another message similar to the following is displayed.

```
Reading c:\xact\data\xdm.pro...
```

This indicates that it is configuring itself by reading the profile information. This configures the Design Manager software as well as the graphics, I/O ports, and any other configurable system parameters. Custom configuration of the xdm.pro file is allowed and is discussed in the Profile menu section.

When XDM has completed loading, your monitor displays the XDM menu screen as shown in Figure 2-1. The menu screen consists of a menu bar at the top, a command line at the bottom, a status area above the command line, a mouse cursor, and a logo with software version information in the center.



```
DesignEntry Translate Fitter Verify Utilities Profile Quit

                         Version: 5.0.0
                Copyright 1989-1993 Xilinx Inc.
    386|DOS-Extender 4.1 - Copyright (C) 1986-1993 Phar Lap Software, Inc.

                        Press F1 for Help

   Family: XC7300
Directory: C:\XACT\TUTORIAL
     Part: 7354-10PC68
    Mouse: MS Mouse

Cmd: _
```

Figure 2-1   XACT Design Manager Opening Screen (PC)

To quit XDM and return to DOS, select Quit on the menu bar at the top of the screen or type **`quit`** on the command line at the bottom of the XDM screen.

This returns you to your system command prompt.

You can temporarily return to DOS without quitting XDM by using the DOS command in the Utilities menu or typing **`dos`** at the command line. Use of this and other utility commands is described later in this chapter.

Caution: If your menus do not display all of the commands included with your Xilinx software product, check your autoexec.bat file to verify that your PATH is properly set to include the executable directory path(s) specified during installation. For example:

```
PATH ...;drive:\XACT;...
```

If it is not present, modify your autoexec.bat file to include it, then reboot your system. The proglist.xdm file may also need to be updated. Do this by invoking XDM and selecting the ScanDisk command from the Utilities menu.

Using XDM on a Workstation

When using XDM on a workstation, X-Windows must first be running. This section acquaints you with the X-Windows environment and then the process of invoking the XACT Design Manager.

About X-Windows and Graphic Interfaces

Xilinx software requires X-Windows to operate on a workstation. The recommended graphic interface is Motif, although Xilinx software is also compatible with Display Manager on HP and Openwindows on Sun-4. It is not compatible with Sunview.

X-Windows is an industry-standard windowing environment developed by the Massachusetts Institute of Technology (MIT). The appearance of the windows, mechanisms used to manipulate windows, and the mouse definition are part of the graphic interface (for example, Motif). X-Windows and Motif are available on HP and Sun-4.

Both X-Windows and the graphic interface are configurable. Mouse operation, menu contents, screen display, and window appearance are controlled by configuration files. The X-Windows environment is controlled by the .Xdefaults file and the Motif window manager is controlled by the .mwmrc file. These files should be located in your home directory.

There are some basic operations that are applicable to most configurations. These items are described in the following paragraphs.

Mouse Configuration

The mouse buttons may be programmed to invoke menus, select objects, and select text. The modes of the mouse button may vary depending upon the location of the cursor. Buttons will operate differently if the cursor is located in an X-terminal window, X-terminal banner or edge, or outside an X-terminal window. Specific operation is dependent upon your configuration.

Window Operations

The size and location of windows are easily modified within the X-Windows environment. Window operations are easily invoked through menus or by the use of accelerators. The basic window operations are described as follows.

- *Move* — This allows you to move the window to a different location on the screen.
- *Size* — This allows you to alter the size of the window by clicking an edge or corner of the window and dragging the cursor.
- *Iconize/Minimize* — This transforms the window into an icon. Double clicking on the icon will restore the window to its original size.
- *Front* — This brings the window to the front of the display, overlapping any other open windows.
- *Back* — This sends the window to the back of the display, allowing all other open windows to overlap it.
- *Pop* — This toggles the window between the front and back of the display. When you initially select Pop, the window comes to the front, and then it toggles the window from front to back and from back to front.
- *Close* — This closes the window and removes it from the active display.

Window Buttons

Many window configurations are defined with three buttons in the banner of the window. These buttons are menu select, iconize, and maximize. The menu select is located in the top left corner and the window iconize and maximize are located in the top right corner.

Window Shortcuts

You can use window shortcuts to resize, move, and front windows. At the command line, simultaneously press the alt key and a function key. On Sun platforms the alt key is actually the meta 〈◊〉 key, located on either side of the space bar. To resize a window using the mouse, click and drag the window edge or corner. To move a window, click and drag on the window banner. To bring a window to the front, click on the banner.

Active Window

The active window is usually highlighted to allow easy identification. The default Xilinx method of choosing the active window is to move the cursor into the window. Other configurations may have other methods. For example, you might have to click in the window to activate it for input.

For more information about your graphic interface, refer to the reference manuals that came with your system.

Starting XDM

To invoke XDM, enter the following command at your operating system command prompt using capital letters.

```
XDM
```

Once the XDM command is entered, the Design Manager software loads into memory and reads the proglist.xdm file to set up your menus. While this is being done, a message similar to the following is displayed.

```
Reading proglist.xdm...
```

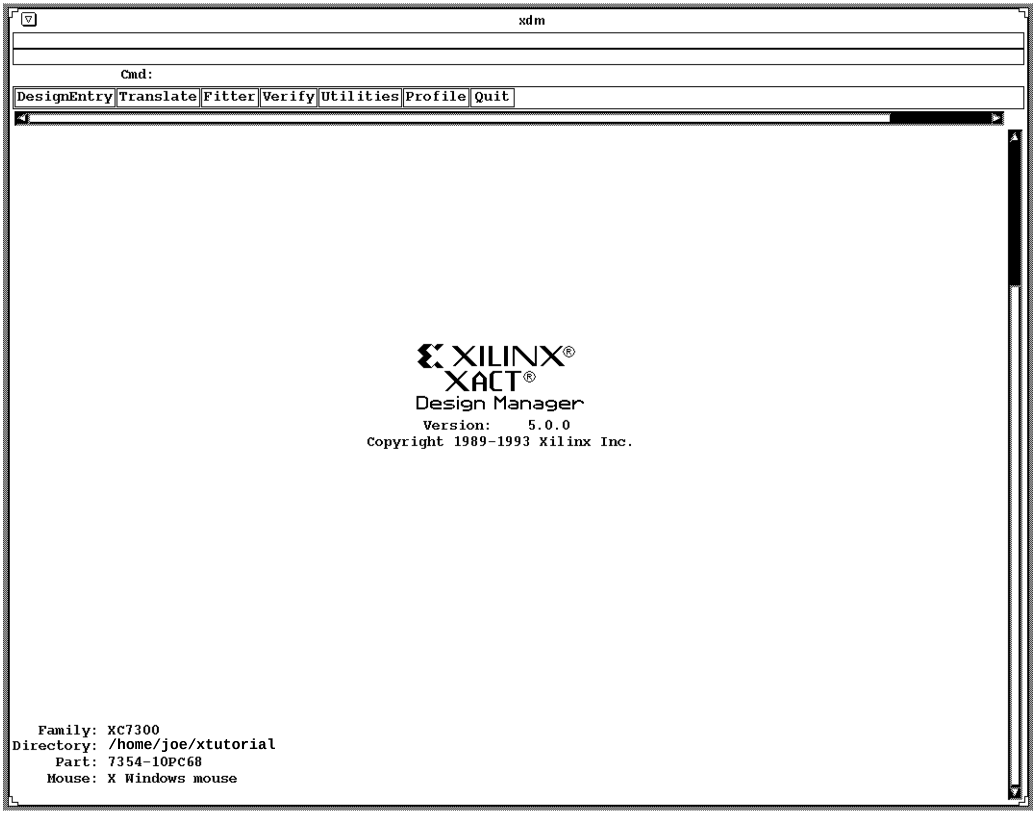
If proglist.xdm is missing, as in the very first invocation of XDM, it is automatically generated by XDM. See the ScanDisk entry in “The Utilities Menu” section in this chapter.

After a short while, another message similar to the following is displayed.

```
Reading xdm.pro...
```

This indicates that XDM is configuring itself by reading the profile information. This configures the Design Manager software as well as the graphics, I/O ports, and any other configurable system parameters. To configure the xdm.pro file, use the Saveprofile command, which is discussed in the Profile Menu section.

When XDM has completed loading, the XDM window appears as shown in Figure 2-2 and consists of four areas.



```
Version:    5.0.0
Copyright 1989-1993 Xilinx Inc.
```

```
   Family: XC7300
Directory: /home/joe/xtutorial
     Part: 7354-10PC68
    Mouse: X Windows mouse
```

Figure 2-2 XDM Opening Screen (Workstation)

- *Status Window* — Starting at the top, there is a status window displaying responses to the commands entered in the command box. Scroll bars allow you to see all of the display.
- *Command Window* — Below the status window is the command window, which contains two areas. The top area displays a history

of previous commands entered at the command line. Keyboard arrow keys or scroll bars can be used to cycle through the commands. Clicking on the command in the history area brings the command down to the command line. The lower area contains the Cmd: prompt, where commands are entered. Commands cannot be entered unless the cursor is blinking on the command line. If the cursor does not appear, click on the command line with the mouse. Commands on the command line can be edited using the backspace and arrow keys.

- *Menu Bar* — Below the command window is the menu bar. Clicking on the menu bars bring up the various menu options.
- *Main Screen* — Below the menu bar is the main screen that contains the version number as well as information on the Xilinx device family selected, the current directory, the part type specified, and the mouse type used.

To quit the XACT Design Manager, select Quit on the menu bar at the top of the screen or enter a **`quit`** command on the command line. This returns you to your system command prompt.

When running your Xilinx software in X-Windows, you can return to a system command prompt by simply opening another window.

Caution: On workstations, if your menus do not display all of the commands included with your Xilinx software product, verify that your path and your environment variables are set correctly. The proglist.xdm file may also need to be updated. Do this by invoking XDM and selecting ScanDisk from the Utilities menu.

Entering Commands

There are two methods for executing each of the commands available in the XDM menu. One method is to use the mouse pointer to open the menu and click on the command you want (the graphic interface). The second method is to enter the command and any specified options from the command line.

Regardless of the method you use to execute commands, messages from commands appear in the window from which you invoked XDM. While a command is executing, the cursor is a clock. When a command finishes executing, a beep sounds, and the cursor changes back to an arrow (or whatever it is set to; see the Cursor command description under “The Profile Menu” section).

The Graphic Interface

Using the graphic interface to execute commands or programs requires that you first know where that command resides in the XDM menu structure.

You also need to know which operations the mouse buttons are set to perform. The usage descriptions provided here assume the mouse is set to the default button configuration. The default is Select, Menu, and Done for B1 (the left mouse button), B2 (the middle mouse button) and B3 (the right mouse button), respectively.

To open a menu, simply position the cursor on the menu you want to open and click the left mouse button. You can also recall the last command selected by pointing anywhere except the menu titles and pressing the middle button.

The Command Line Interface

The command line is used for entering commands listed in the menus from the keyboard and is displayed at the bottom or top or bottom of your screen as follows.

```
Cmd:
```

If a command is displayed in a menu with two or more capital letters, the capital letters represent a shortcut command entry.

For example, the Utilities menu displays a command called DirClean. In this case, you can type three different entries on the command line: DirClean, DirC, or DC, followed by a return to execute the command.

When a command line shortcut is used to execute a command, the case of the characters is not significant. In the preceding example, entering a dc, Dc, or dC would provide the same result as entering DC.

On PC systems, when you open either the Utilities or Profile menu, notice that the commands are displayed with some or all of the characters highlighted. The highlighted characters represent an additional shortcut for entering the command on the command line.

Task-Based Command Flows

The XDM menu contains all the commands you need to turn your schematic or behavioral design files into device programming files and simulation files. Although there are standard sequences of commands for common tasks, such as creating a design database or preparing for timing simulation, the commands are displayed separately to allow for maximum flexibility. This section shows standard command sequences you need to use to accomplish the most common tasks.

The options for these commands are not needed for the commands to work. When prompted for an option, you can select **Done** to bypass all options.

For more details on each command, see the "Menu Tour" section.

Schematic Design to Programming File

To create an Intelhex programming file from a schematic design, follow these steps:

1. Use the **Workview** or **OrCAD** command on the **DesignEntry** menu (you can also develop Mentor Graphics or CADENCE designs under their respective interfaces). Create the schematic using XC7000 library components. See the *Interface User Guide* for your CAE tool for more details.
2. Create any PLUSASM equation files needed to describe logic under PAL symbols. You can translate source files from other PLD compiler tools or from PAL JEDEC files. See the *Interface User Guide* for your CAE tool for more details.
3. Use the **XEMAKE** command on the **Translate** menu.
4. Select the **Make Intelhex Bitmap** option of XEMAKE.

To create a JEDEC programming file from a schematic design, follow these steps:

1. Use the **Workview** or **OrCAD** command on the **DesignEntry** menu. Create the schematic using XC7000 library components. See the *Interface User Guide* for your CAE tool for more details.
2. Create any PLUSASM equation files needed to describe logic under PAL symbols. See the *Interface User Guide* for your CAE tool.

3. Use the **XEMAKE** command on the **Translate** menu.
4. Select the **Make Design Database** option of XEMAKE.
5. Use the **MAKEJED** command on the Verify menu.

About the XEMAKE Command

The XEMAKE command runs several other commands, which you can run separately if you wish.

XEMAKE examines the dates of all the files in your design (.pld equation files, lower-level schematics, and so on) and attempts to process only the files that have changed since you last ran XEMAKE on your design.

If you select a schematic file as the input file, the XEMAKE command flow is as follows:

1. Checks the schematic (Viewlogic only).
2. Runs **WIR2XNF** or **SDT2XNF** (Translate menu) on the schematic file.
3. Runs **PLUSASM** (Translate menu) on the PLUSASM equation file(s).
4. Runs **XNFMERGE*** (Translate menu) to merge the design files.
5. Runs **FITNET**** (Fitter menu) to fit the design and map it to an EPLD device.
6. Runs **MAKEPRG** (Verify menu) to create an Intelhex programming file.

* You have the option of starting the XEMAKE flow just before XNFMERGE, using .xnf files as input. Do this to save time if none of your schematics have changed. To do this, use the -X option.

** You have the option of stopping the XEMAKE flow just after FITNET, which produces a .vmh (or .vmd) file you can use to create either an Intelhex (with MAKEPRG) or JEDEC (with MAKEJED) programming file. To do this, choose “Make Design Database” as the target.

If you select a behavioral design file (.pld or .pds) as the input file, the XEMAKE command flow is as follows:

1. Runs **FITEQN**** (Fitter menu) to fit the design and map it to an EPLD device.
2. Runs **MAKEPRG** (Verify menu) to create an Intelhex programming file.

** You have the option of stopping the XEMAKE flow just after FITEQN, which produces a design database (.vmh) file you can use to create either an Intelhex (with MAKEPRG) or JEDEC (with MAKEJED) programming file. To do this, choose “Make Design Database” as the target.

Behavioral Design to Programming File

To create a programming file from a design that is a set of separate PAL equation files, follow these steps:

1. Create the PAL equation files.
2. Use the **PALCONVT** command on the **Fitter** menu.
3. Select the **Create New PLD and PAL Interconnect Report** option of **PALCONVT**.
4. Check the top-level *design_name*.pld file to make sure that the input and output signals were correctly classified as INPUTPIN, OUTPUTPIN, or NODE. If the signals were incorrectly classified, edit the *design_name*.pld file.
5. Use the **XEMAKE** command on the **Translate** menu or the **FITEQN** command on the **Fitter** menu.
6. Select the **Make Intelhex Bitmap** option of XEMAKE or use the **MAKEPRG** (for an Intelhex file) or **MAKEJED** (for a JEDEC file) command on the Verify menu.

If you are modifying an existing design and you are sure the input and output signals will be correctly classified, follow these steps:

1. Modify the PAL equation files.
2. Use the **PALCONVT** command on the **Fitter** menu.
3. Select the **Integrate New PLD Using FITEQN** option of **PALCONVT**.
4. Use the **MAKEPRG** (for an Intelhex file) or **MAKEJED** (for a JEDEC file) command on the **Verify** menu.

To create a programming file from a design that is all in a single file or that uses INCLUDE_EQN statements to link files, follow these steps:

1. Create the equation files.
2. Use the **XEMAKE** command on the **Translate** menu or the **FITEQN** command on the **Fitter** menu.
3. Select the **Make Intelhex Bitmap** option of XEMAKE or use the **MAKEPRG** (for an Intelhex file) or **MAKEJED** (for a JEDEC file) command on the Verify menu.

JEDEC PAL File to PLUSASM PAL File

If you have a JEDEC file that maps a design for a 20V8 or 22V10 PAL, you can convert it to a PLUSASM file using the **JED2PLD** command on the **Translate** menu. Then use XEMAKE or FITEQN (if the PAL is part of a purely behavioral design) or FITNET (if part of a schematic design) to fit the design.

Workview Simulation

This section describes the steps necessary to perform functional or timing simulation in Viewlogic. You can also perform timing simulation in Viewlogic if you have a behavioral design.

Functional Simulation

To perform functional simulation in Viewlogic, follow the steps below. Perform all steps within the Viewlogic environment.

1. Create your design in Viewdraw using XC7000 library components.
2. Select the **Export** → **Wirelist** → **Viewsim** command or type **`vsm`** at the command line.
3. Perform simulation in Viewsim using the standard procedure. For more information, refer to the *Viewlogic Interface User Guide*.

Note: You can perform functional simulation only if your design consists entirely of schematic library components and contains no equation files.

Timing Simulation

To perform timing simulation in Viewlogic, follow the steps below.

1. Create your design in Viewdraw using XC7000 library components.
2. Exit Viewlogic and enter or return to XDM.
3. Use the **Translate → XEMAKE** command or the equivalent set of commands to fit your design.
4. Select the **Verify → XSIMMAKE** command.
5. Choose **Viewlogic_Epld_Timing** as the target.
6. Exit XDM and enter or return to Viewlogic.
7. Perform simulation in Viewsim using the standard procedure. For more information, refer to the *Viewlogic Interface User Guide*.

Timing Simulation for a Behavioral Design

To perform timing simulation in Viewlogic for a behavioral design, follow the steps below.

1. Create your design.
2. Use the **Translate → PALCONVT** or **Translate → FITEQN** command to fit your design.
3. Select the **Verify → XSIMMAKE** command.
4. Choose **Viewlogic_Epld_Timing** as the target.
5. Enter Viewlogic.
6. Perform simulation in Viewsim using the standard procedure. For more information, refer to the *Viewlogic Interface User Guide*.

OrCAD Simulation

This section describes the steps necessary to perform functional or timing simulation in OrCAD. You can also perform timing simulation in OrCAD if you have a behavioral design.

Functional Simulation

To perform functional simulation in OrCAD, follow the steps below.

1. Create your design in Draft using XC7000 library components.
2. Exit OrCAD and enter or return to XDM.
3. Select the **Translate** ➝ **ANNOTATE** command.
4. It is recommended that you select the **Translate** ➝ **CLEANUP** command to detect any schematic drawing errors.
5. Select the **Translate** ➝ **INET** command to generate an OrCAD INF-formatted netlist.
6. Select the **Verify** ➝ **XSIMMAKE** command.
7. Choose **OrCAD_Epld_Func** as the target.
8. Exit XDM and enter or return to OrCAD.
9. Perform simulation using the OrCAD simulation tool (VST). The netlist file created by XSIMMAKE is *design_name*.vst. For more information, refer to the *OrCAD Interface User Guide*.

Note: You can perform functional simulation only if your design consists entirely of schematic library components and contains no equation files.

Timing Simulation

To perform timing simulation in OrCAD, follow the steps below.

1. Create your design in Draft using XC7000 library components.
2. Exit OrCAD and enter or return to XDM.
3. Use the **Translate** ➝ **XEMAKE** command or the equivalent set of commands to fit your design.
4. Select the **Verify** ➝ **XSIMMAKE** command.
5. Choose **OrCAD_Epld_Timing** as the target.
6. Exit XDM and enter or return to OrCAD.
7. Perform simulation using the OrCAD simulation tool (VST). The netlist file created by XSIMMAKE is *design_name*.vst. For more information, refer to the *OrCAD Interface User Guide*.

Timing Simulation for a Behavioral Design

To perform timing simulation in OrCAD for a behavioral design, follow the steps below.

1. Create your design.
2. Use the **Translate → PALCONVT** or **Translate → FITEQN** command to fit your design.
3. Select the **Verify → XSIMMAKE** command.
4. Choose **OrCAD_Epld_Timing** as the target.
5. Enter OrCAD.
6. Perform simulation using the OrCAD simulation tool (VST). The netlist file created by XSIMMAKE is *design_name*.vst. For more information, refer to the *OrCAD Interface User Guide*.

About the XSIMMAKE Command

The XSIMMAKE command runs several other commands, which you can run separately if you wish. The XSIMMAKE command flow is different for each target.

- For the **Viewlogic_Epld_Timing** target, XSIMMAKE runs **VMH2XNF** (Verify menu) on the .vmh file, **XNF2WIR** (Verify menu) on the .xnf file, then **VSM** (Verify menu) on the WIR file.
- For the **OrCAD_Epld_Func** target, XSIMMAKE runs **SDT2XNF** and **XNFMERGE** on the .inf file from INET, then runs **XNF2VST** (Verify menu) on the .xnf file.
- For the **OrCAD_Epld_Timing** target, XSIMMAKE runs **VMH2XNF** (Verify menu) on the .vmh file, then runs **XNF2VST** (Verify menu) on the .xnf file.

Menu Tour

You can get acquainted with XDM by taking a tour of the menus and the functions they perform. This section includes general information concerning the menus that may be displayed by XDM for XEPLD software.

First, the Settings Fields are described. You determine the most important aspects of your design environment using these fields.

The menu commands described in this chapter are visible only after you have selected XC7200 or XC7300 as the Family. Depending on which Xilinx software you have installed, your system may actually display more or fewer items than those listed and described here. For example, some commands only appear if you have installed DS35, the Xilinx OrCAD interface, or DS391, the Viewlogic interface.

After each menu description, the commands on that menu are described. For the commands that have a DOS or UNIX equivalent, the DOS or UNIX command syntax is also described. The command syntax for DOS and UNIX is identical except where noted.

After a general description of each of the possible menus, more detailed information is provided for the menus specifically supplied by XDM software. These are the Utilities and Profile menus.

To open a menu, place your mouse pointer on the menu you want to open and click the left mouse button. In each of the menus you will find a list of commands that are available.

The Settings Fields

Four fields in the lower left corner of the XDM screen determine some of the most important settings for your design environment:

- ***Family*** — Specifies the family of Xilinx devices that are the target for your design. To choose EPLD devices, select **XC7200** or **XC7300**.

  This field determines many of the menu options that are available. For example, the PLUSASM command appears on the Translate menu only if you choose XC7200 or XC7300.
- ***Directory*** — Specifies the directory in which you are working. This is the same as the Directory command on the Utilities menu.
- ***Part*** — Specifies the part that is the target of your design, for example, XC7236-30PC44. You can also specify “InDesign” as the part type; this tells the software to look for a PART= attribute in a schematic design or at the CHIP statement in a behavioral design.

  If you are typing a command on the command line and do not supply a part type, the default is the fastest part available, the XC73108-10PC84.

If you supply an abbreviated part type on the command line, the default is the largest and fastest chip of that type, as follows:

73108	XC73108-10PC84
7372	XC7372-10PC84
7354	XC7354-10PC68
7336	XC7336-5PC44
7318	XC7318-5PC44
7272	XC7272-25PC84
7272A	XC7272A-16PC84
7236	XC7236-25PC44
7236A	XC7236A-16PC44

- ***Mouse*** — Specifies mouse mode (or connection port on the PC).

Note: To save a customized profile, including family and part settings, mouse type, and so on, use the Saveprofile command in the Profile menu. This allows you to use the same settings automatically each time you start up XDM.

The Design Entry Menu

This menu contains a listing of the design entry software packages installed in your system. When you select any of the commands listed in this menu, XDM starts up the design entry software and you can create your design.

The Translate Menu

This menu contains all programs required for producing a netlist. The following bullet items are brief descriptions of the various translate programs.

XEMAKE (EPLD Automatic Implementation Tool)

Follow these steps to use the XEMAKE command:

1. Select an option if you wish:
   - -G — Creates a makefile and stops. You can run this makefile later to complete the command. Running the makefile is faster than selecting the schematic, because the makefile stores information that the software otherwise has to compute (such as the names and locations of PLD files) if you select the schematic.

- -R — Forces execution of all design files regardless of file dates, and displays makefiles in the input file list along with the schematic and behavioral design files.
- -X — Prompts for an XNF file instead of a schematic or behavioral design file. Looks in the xnf directory before the current directory. This is useful if you want to fit a design in an XNF file from a third-party tool to an EPLD device.

  You can select one of these options or none. If you select no options, this command prompts for a schematic or behavioral design file, does not stop after makefile creation, and looks at file dates to determine which files were modified since you last ran XEMAKE on your design. After you have selected any options you want (or no options), select the Done button.

2. Next, select the top-level schematic file, makefile (.mak), or behavioral design file (.pld or .pds). If you have used XEMAKE before, you will see one or more makefiles listed. Select the file you want followed by the Done button.
3. Select the target from these alternatives:
   - Make Intelhex bitmap
   - Make design data base

   You can also create a bitmap file later using the MAKEPRG or MAKEJED command if you choose "Make design data base".
4. If you chose to create an Intelhex bitmap, type a signature in response to the `Enter signature:` prompt. A signature is a series of letters or numbers, up to 8 characters long with a ".A" extension, that indicates the revision of the design. The device programmer can read the signature, and the person running the device programmer can verify that the version is correct.

Use this command to fit a design using a single command. For a schematic design, this command runs WIR2XNF or SDT2XNF, runs XNFMERGE, processes any PAL files using PLUSASM, runs FITNET, and optionally runs MAKEPRG. The PAL files must have .pld or .pds extensions. For a behavioral design, this command runs FITEQN and optionally runs MAKEPRG.

Unless you use the -R option, the XEMAKE command regenerates the netlist only if the schematic has changed since the design was last fitted. This command reassembles the PAL files only if they have changed since the design was last fitted.

The XEMAKE command does not run JED2PLD, ABL2PLD, PALCONVT, PINSAVE, or MAKEJED.

DOS or UNIX command equivalent:

xemake -g -r -x -p *part_type design.ext target***.prg**

The -g, -r, and -x options are described above.

The -p option allows you to specify an XC7000 device. If you omit the -p option, the software looks for a PART= attribute in a schematic design or at the CHIP statement in a behavioral design file. If the software finds no part type, the default is the fastest and largest part, the XC73108-10PC84.

If you want to create an Intel HEX file, specify a target file name with a .prg extension.

You can then run XEMAKE again to execute the makefile:

xemake -r *design_name.***mak**

ABL2PLD (Create EPLD from Xilinx ABEL Design)

Translates a .abl file into a PLUSASM (.pld) file. If the file represents a PAL in a schematic design, ABL2PLD runs the PLUSASM command on the .pld file. If the file is a stand-alone behavioral design or the top-level file of a stand-alone design, ABL2PLD runs the FITEQN command. You indicate that the file represents a stand-alone design by using the -R option.

ANNOTATE (Annotate SCH — PC Only)

Updates reference designators in OrCAD schematics. This is the first step in preparing an OrCAD schematic for functional simulation.

CLEANUP (Cleanup SCH — PC Only)

Cleans up overlapping objects in OrCAD schematics.

INET (Compile SCH — PC Only)

Creates a .inf file as the second step in preparing an OrCAD schematic for functional simulation.

SDT2XNF (Create Orcad Netlist — PC Only)

Use this command to generate a netlist from an OrCAD SDT schematic file. Follow these steps to use the SDT2XNF command:

1. Select a .inf file from the displayed list.
2. Select an output .xnf file from the displayed list, or select New File to create a new output file.
3. Select an option if you wish:
   - -D *directory* — Writes output files to the specified directory.
   - -S path — Specifies a search path for Xilinx-defined .inf files.
   - -U path — Specifies a search path for user-defined .inf files.

   You can select one of these options or none. If you select no options, the output files are written to the xnf directory below the current directory. After you have selected the options you want, select the Done button.

After using this command, use the XNFMERGE command, which creates a .xff file for input to the XEPLD Fitter.

Do not use SDT2XNF to prepare an OrCAD schematic for functional simulation. You must use XSIMMAKE to do this.

DOS or UNIX command equivalent:

`sdt2xnf` *design_name*

WIR2XNF (Create Workview Netlist)

Use this command to generate a netlist from a Viewlogic schematic. Follow these steps to use the WIR2XNF command:

1. Select an option if you wish:
   - -B — Switches off status lines.
   - -C — Checks pin-to-pin and pin-to-block connections.
   - -F — Flattens the design hierarchy into one file. The default is to create one .xnf file for each level of the hierarchy.
   - -OD *directory* — Writes output files to the specified directory.
   - -X — Maintains the hierarchy except for Xilinx macros. If you

use this option, you must use the -F option as well.

You can select one of these options or none. If you select no options, status lines are on and the output files are written to the current directory. After you have selected the options you want, select the Done button.

2. Select a top-level schematic file from the displayed list.
3. Select an output .xnf file from the displayed list, or select New File to create a new output file.

After using this command, use the XNFMERGE command, which creates a .xff file for input to the XEPLD Fitter.

DOS or UNIX command equivalent:

```
wir2xnf design_name
```

JED2PLD (Import & Assemble PLD JEDEC)

Imports a JEDEC file to define the functional behavior of a PLD component in a schematic. JED2PLD first translates the file specified by JEDEC File to a PLUSASM equation file (with a .pld extension), then subsequently invokes the PLUSASM assembler, which assembles the equation file into a bitmap.

You can view and edit the PLUSASM equation file. If you edit the PLUSASM file, you must use the PLUSASM command to recompile the PLUSASM file. XEMAKE invokes PLUSASM automatically.

DOS or UNIX command equivalent:

```
jed2pld -e equation_file -p part_type -j jedec_file
   -t pld_type
```

The *equation_file* is the name of the .pld file you are creating; the *jedec_file* is the JEDEC file you are reading. The *pld_type* is one of the supported PLD types (PL20V8 or PL22V10).

PLUSASM (Assemble PLD Equations)

Assembles a PLUSASM equation file that describes a PLD used in a schematic design. PLUSASM assembles the source file you select and generates a component bit-map file and a report. Component bit-map files are used by the XEPLD Fitter commands on the Fitter menu.

DOS or UNIX command equivalent:

```
plusasm -e equation_file -p part_type
```

If you omit the -p option, the software looks at the CHIP statement in the equation file. If the software finds no part type, the default is the fastest and largest part, the XC73108-10PC84.

PINSAVE (Save EPLD Pinout)

Use this command after a successful fitting of your design to save the pin allocation information into a Pin-save file with extension .vmf. If you set the Pinfreeze option of the FITEQN or FITNET command to On, the Fitter will, on subsequent iterations of your design, assign the pins to the same locations indicated in this file.

DOS or UNIX command equivalent:

```
pinsave -n design_name
```

If your design is targeted at an XC7272 device, you must specify the .vmd extension after the design name.

XNFMERGE (Merge Multiple XNF Files)

This command merges XNF files generated by the SDT2XNF or WIR2XNF command. Follow these steps to use the XNFMERGE command:

1. Select an option if you wish:

   - -A — Abbreviates the file-reading report.
   - -D *directory* — Adds the specified directory to the .xnf file search path.

     Note: If you specified -D for SDT2XNF or -OD for WIR2XNF, be sure to specify -D for XNFMERGE.
   - -I — Ignores RLOC related information (FPGA only).
   - -O *file* — Sends the merge report to the specified file. If you do not use this option, the default merge report name is *design_name*.mrg.
   - -P — Changes the package specification for the part.
   - -Q — Suppresses messages about unresolvable symbols. This

option is always used automatically for XC7000 designs.

You can select one of these options. Normally you do not need to specify any options for EPLD designs. After you have selected the options you want, select the Done button.

2. Select a .xnf file from the displayed list. If there is an xnf directory below the current directory, XDM looks there for files to put in this list.

3. Select an output .xff file from the displayed list, or select New File to create a new output file.

DOS or UNIX command equivalent:

`xnfmerge` *design_name*

The Fitter Menu

Commands on this menu execute the XEPLD design implementation algorithms that map the design onto the target EPLD device.

FITEQN (Integrate EPLD Behavioral Design)

Use this command to fit a behavioral design. Follow these steps to use the FITEQN command:

1. Select the top-level equation file from the list of .pld and .pds files. For information about creating a hierarchy of design files, see the *XEPLD Design Guide*.

2. Select an option if you wish:

   - -i — Ignores the pin assignments in the equation file.
   - -u — Drives unused I/O pads, clock signals, FOE signals, and CE pins to GND.

3. If you have previously saved pin allocation information in a .vmf file (using the PINSAVE command), the FITEQN command presents the -f (Pin-freezing on) option, which allows you to assign pins to the same positions with each iteration of your design. The -f option is always off by default. Selecting the -f option repeatedly before you select Done toggles the -f option on and off. The on or off setting of this option is displayed in a status line at the top or bottom of the XDM screen just above the command line.

The -i and -f options are mutually exclusive. For more information, see Table 2-1.

The main equation file you specify must follow the required PLUSASM file structure. The equation file is processed by several XEPLD modules to produce a database. From this database, a programming file can be produced to program the device.

This command produces the Resource (.res), Mapping (.map), Pinlist (.pin), Partitioner (.par) and Logic Optimizer (.lgc) reports. It also produces a log file (.log) and a behavioral design file (.eqn) that shows the optimized and placed design in PLUSASM format. For more details about reports, refer to the “Report Formats” chapter.

If you change the pin assignments in the equation file, these assignments override the assignments in the .vmf file.

DOS or UNIX command equivalent:

fiteqn -e *equation_file* **-p** *part_name* **-u -f | -i**

If you omit the -p option, the software looks at the CHIP statement in the top-level behavioral design file. If the software finds no part type, the default is the fastest and largest part, the XC73108-10PC84.

FITNET (Integrate EPLD Schematic Design)

Use this command to fit a schematic design. Follow these steps to use the FITNET command:

1. Select the design file from the list of .xff files.
2. Select an option if you wish:
   - -i — Ignores the pin assignments in the schematic.
   - -u — Drives unused I/O pads, clock signals, FOE signals, and CE pins to GND.
3. If you have previously saved pin allocation information in a .vmf file (using the PINSAVE command), the FITNET command presents the -f (Pin-freezing on) option, which allows you to assign pins to the same positions with each iteration of your design. The -f option is always off by default. Selecting the -f option repeatedly before you select Done toggles the -f option on and off. The on or off setting of this option is displayed in a status line at the top or bottom of the XDM screen just above the command line.

The -i and -f options are mutually exclusive. For more information, see Table 2-1.

The input .xff file is processed by several XEPLD modules to produce a database. From this database, a programming file can be produced to program the device.

This command produces the Resource (.res), Mapping (.map), Pinlist (.pin), Partitioner (.par) and Logic Optimizer (.lgc) reports. For more information about reports, refer to the “Report Formats” chapter.

DOS or UNIX command equivalent:

fitnet -n *design_name* **-p** *part_name* **-u -f | -i**

If you omit the -p option, the software looks for a PART= attribute in the schematic file. If the software finds no part type, the default is the fastest and largest part, the XC73108-10PC84.

Table 2-1 Interaction of -f and -i FITEQN and FITNET Parameters

	-i off	**-i on**
-f off	PINSAVE file **ignored**. Schematic or equation file pin assignments **used**.	PINSAVE file **ignored**. Schematic or equation file pin assignments **ignored**.
-f on	PINSAVE file pin assignments **used**. Schematic or equation file pin assignments **override** those in the PINSAVE file if they are achievable.	The -f and -i options are mutually exclusive.

PALCONVT (Convert PAL-based Design)

This command creates a new design that includes existing PALs in the form of .pds or .pld files. It is a preliminary step to using FITEQN that saves you from having to create a top-level file for a design made up of PAL files. Follow these steps to use the PALCONVT command:

1. Type a name for the new top-level file after the following prompt:

```
Enter design file name (.pld):
```

If you enter the name of an existing .pld file, you are asked if you want to overwrite that file. You may want to overwrite your top-level file if you are using PALCONVT a second time with the same set of PAL files.

2. Select the PAL files that make up your design from the displayed list of .pld and .pds files.

   These files are PLUSASM-compatible files (PALASM Boolean equation files) that you created with a text editor or a PAL compiler such as ABEL or CUPL.

3. Select the target from these alternatives:

   - Create new PLD and PAL Interconnect Report
   - Integrate new PLD using FITEQN

   The first alternative lets you stop after creating the top-level file and then use FITEQN later. This allows you to make changes to the design (for example, assigning pins or changing pin types) by editing the top-level file.

The PALCONVT command reads the PAL file equations, resolves any polarity inversions, resolves any PAL-specific functionality, and automatically determines all external requirements for dangling signals.

The *design_name*.pld file that PALCONVT creates contains pin declarations, NODE declarations, and INCLUDE_EQN statements for the design. You can edit this file to specify additional external signals, for example by redeclaring nodes as output pins.

The PALCONVT command also generates a PAL Interconnect report named *design_name*.int, which summarizes the number of equations found and I/O pins created. You can use this report to help you choose the best target device.

After using PALCONVT, choose a target device for your converted design, then use the FITEQN command to fit your design.

DOS or UNIX command equivalent:

palconvt -f -p *part_name design_name PAL_file1 PAL_file2..*

You do not need to specify the .pld extension on the *design_name* or the .pld or .pds extensions on each *PAL_file*. The -f option runs FITEQN if PALCONVT ran without errors.

If you omit the -p option, the software looks at the CHIP statement in the top-level behavioral design file. If the software finds no part type, the default is the fastest and largest part, the XC73108-10PC84.

If your design contains so many PAL files that the PALCONVT command as typed on the command line exceeds 128 characters, you can type the names of the PAL files in an ASCII file (called a PAL list file) and use the following variation of the PALCONVT command:

`palconvt -f -p` *part_name design_name @PAL_list_file*

The Verify Menu

The Verify menu provides a selection of programs associated with design simulation and device programming. These selections include all simulation and verification programs and all utility programs needed to create the required file formats.

XSIMMAKE (Create Simulation File)

This command performs all the steps necessary to create a simulation netlist file for functional (OrCAD only) or timing (OrCAD or Viewlogic) simulation. (Viewlogic functional simulation is performed entirely within the Viewlogic environment.) Follow these steps to use the XSIMMAKE command:

1. Select an option if you wish:
   - -O — Specifies the name of the output simulation netlist file, minus the extension; the default is the input file name. If you perform functional simulation, you may wish to use a different name for your timing simulation netlist file.
2. Select the target from these alternatives:
   - Viewlogic_Epld_Timing — Prepares a timing simulation netlist file (VSM format) based on a completed EPLD design (VMH database); runs VMH2XNF, XNF2WIR, and VSM.
   - OrCAD_Epld_Func — Prepares a functional simulation netlist file (with .vst extension) for an entirely schematic-based design (no equations); runs SDT2XNF, XNFMERGE, and XNF2VST.
   - OrCAD_Epld_Timing — Prepares a timing simulation netlist file (with .VST extension) based on a completed EPLD design (VMH database); runs VMH2XNF and XNF2VST.

3. Select the design file name from the displayed list of .vmh files.

DOS or UNIX command equivalent:

xsimmake -f *target* **-o** *output_directory*

Choosing a target and using the -o option are described above.

ASCTOVST (OrCAD/VST Utility — PC Only)

Converts an OrCAD Stimulus or Trace file from ASCII to binary format or vice versa. The ASCTOVST utility is described in the *OrCAD Digital Simulation Tools* manual.

ORCAD (OrCAD VST — PC Only)

Puts you in the OrCAD/ESP design environment, where the OrCAD VST simulator is located.

XNF2VST (Create Orcad Sim Files — PC Only)

This command creates a timing simulation model file of a completed XEPLD design for use with OrCAD's VST simulator. Follow these steps to use the XNF2VST command:

1. Select a .xnf file from the displayed list that has been generated by VMH2XNF. (Do not select a .xnf file generated by SDT2XNF. Such a file cannot be simulated; you will get warnings about simulation models that XNF2VST cannot find in the library.)
2. Select a .vst file from the displayed list, or select New File to create a new .vst output file.
3. Select an option if you wish:
   - -O — Specifies the name of the output simulation netlist file, minus the extension; the default is the input file name. If you perform functional simulation, you may wish to use a different name for your timing simulation netlist file.
   - -R — Reads the existing .nrf (name reference) file. This file lists the names and name changes of symbols and nets.
   - -U — Specifies that the simulation file will include unit delays only. Select this option if you are performing functional simulation.

- -W — Overwrites existing .ast (ASCII stimulus) and .atr (ASCII trace) files.
- -X *path* — Specifies the XACT path.

You can select one of these options or none. After you have selected the options you want, select the Done button.

DOS or UNIX command equivalent:

`xnf2vst` *design_name*

Note: Use only XSIMMAKE to make a VST netlist file for OrCAD functional simulation.

VSM (Viewsim Wirelister)

This command reads the model file produced by XNF2WIR and creates a Viewsim wirelist file (with a .vsm extension) for timing simulation. You can also invoke the VSM command from either XDM or the Viewlogic menu to prepare for functional simulation of your schematic. Follow these steps to use the VSM command:

1. Select the design file from the displayed list.
2. Select an option if you wish:
   - -D *file* — Creates a delay back-annotation file with a .dtb extension. (FPGA only)
   - -F *file* — Specifies the name of the wirelist output file. The default is the same name as the input file (but with a different extension).
   - -H — Creates a hierarchical netlist. This option is on by default.
   - -S — Creates a short format netlist.
   - -W — Generates a WIR file.

   You can select one of these options or none. (Normally you do not need to select any of these options.) After you have selected the options you want, select the Done button.

DOS or UNIX command equivalent:

`vsm` *design_name*

XNF2WIR (Create Viewlogic Sim Files)

Creates a timing simulation model (wirelist file) of a completed XEPLD design for use with Viewlogic's Viewsim simulator. Follow these steps to use the XNF2WIR command:

1. Select an option if you wish:
   - -A — Specifies the name of the AKA file to use (FPGA only).
   - -B — Switches off status lines.
   - -L — Tags each symbol in the WIR file with the library alias that appears in your viewdraw.ini file.
   - -M *number* — Allows you to specify a maximum number of simulation primitives per output file. The default is 1000. If XNF2WIR runs out of memory, try running the command again using a smaller number for this option (try 50% of the last value used).
   - -R — Only creates the necessary command file to initialize ROMs in the design (FPGA only).

   You can select one of these options or none. After you have selected the options you want, select the Done button.
2. Select a .xnf file from the displayed list that has been generated by VMH2XNF. (Do not select a .xnf file generated by WIR2XNF or XNFMERGE; such a file cannot be simulated.)

DOS or UNIX command equivalent:

`xnf2wir -m` *module_limit design_name*

The "-m *module_limit*" field is optional; it conserves memory by specifying the maximum number of simulation modules per wirelist file.

MAKEJED (Make JEDEC Programming Files)

Creates a file in JEDEC format that you can use to program an XEPLD device. This command prompts you for a signature, which you must specify.

DOS or UNIX command equivalent:

`makejed -n` *design_name* **`-s`** *signature*

If you do not specify the extension in the file name, this command looks for a .vmh file. If the command cannot find a .vmh file, it looks for a .vmd (XC7272) file. If you have a design that has been mapped to both an XC7272 device and another device, you must specify the .vmd extension to use the .vmd file.

The *signature* is a series of letters or numbers, up to 8 characters long with a ".A" extension, that indicates the revision of the design. The device programmer can read the signature, and the person running the device programmer can verify that the version is correct.

MAKEPRG (Make Hex Programming Files)

Creates a file in Intel Hex format that you can use to program an XEPLD device. This command prompts you for a signature, which you must specify.

DOS or UNIX command equivalent:

makeprg -n *design_name* **-s** *signature*

If you do not specify the extension in the file name, this command looks for a .vmh file. If the command cannot find a .vmh file, it looks for a .vmd (XC7272) file. If you have a design that has been mapped to both an XC7272 device and another device, you must specify the .vmd extension to use the .vmd file.

The *signature* is a series of letters or numbers, up to 8 characters long with a ".A" extension, that indicates the revision of the design. The device programmer can read the signature, and the person running the device programmer can verify that the version is correct.

VMH2XNF (Make XNF for Timing Simulation)

Creates a .xnf file with timing parameters for use in timing simulation. The input file can be a .vmh or .vmd (from XC7272) file. Follow these steps to use the VMH2XNF command:

1. Select an option if you wish:
    - -L 4 or -L 5— Allows you to choose the 4.0 or 5.0 Xilinx EPLD library regardless of the library you used to create your design. The default is the 5.0 library for a new design or the 4.0 library for a design created with 4.0 components. You may want to use the -L 4 option if you are using the XNF file with third-party software that has not yet been upgraded to 5.0, for example

logic modeling software.

- -O — Changes the name of the output file, minus the extension. This prevents your timing simulation XNF file from overwriting the XNF file that contains your design.

2. Select a .vmh or .vmd file from the displayed list.

Note: The XNF file produced by this command is written to the current directory by default, not the xnf directory, so it will not overwrite the XNF file produced by WIR2XNF or SDT2XNF unless you change directory and output file options.

This XNF file is different from the XNF file produced by WIR2XNF or SDT2XNF, which is based on the schematic and used for design capture. The XNF file produced by VMH2XNF contains only simulation primitives with actual timing parameters. It contains an image of the EPLD device constructed from the bitmap in the .vmh file. It does not consist of the library symbols used to capture your design and cannot be used as an input file to the software. The names of signals connected to device pins and component outputs are preserved except where component outputs are optimized.

DOS or UNIX command equivalent:

```
vmh2xnf design_name -l -o
```

PROLINK (Program EPLD Device — PC Only)

Invokes the PROLINK interface software for controlling and downloading Hex programming files (generated by MAKEPRG) to the Xilinx DS120 device programmer.

The Utilities Menu

The Utilities menu provides several utility commands to let you change working directories, see which versions of Xilinx–supported software are installed on your system, view file contents, and use other file management and system control features. The commands in the Utilities menu are not related to the development software loaded in your system. They are the same regardless of your software configuration and are described in the following paragraphs.

Browse

This utility allows you to view, as a read–only document, any text file from within the Xilinx Design Manager. After selecting Browse, you are prompted for a file name. To browse through a file, select the file name from the menu or enter it from the command line. Press F1 to return to the Design Manager.

If you wish to have XDM invoke your own text viewing program when the Browse command is selected, you may do so by setting an environment variable in your autoexec.bat or .cshrc file. The variable BROWSE should be set in the autoexec.bat or .cshrc file to the name of the text-viewing program. Consider the following examples:

```
SET BROWSE=LIST           (DOS)
setenv BROWSE more        (UNIX)
```

Browsing a file within the Design Manager can be slow and should only be used if you do not have a text editor on your system. See the Edit command.

DirClean

This provides a method to help you manage your design directories by eliminating unwanted files created by the design translation process. To use DirClean, select the command and then select any files you want to remove from the current directory; the selected files then become highlighted. When you have selected all of the files you want to remove, click on Done or press Enter to execute their removal.

Directory

The Directory command allows you to move easily through your directory structure. This command allows you to select the current directory that XDM reads from and writes to. When selected, a menu appears displaying the current directory, the parent directory, and all subdirectories of the current directory. Each time you select a new current directory, the menu changes to reflect the new parent directory and subdirectories.

To change disk drives on the PC, enter the following command at the command line.

`dir` *drivespec*

In this command line entry, *drivespec* represents the disk drive you want as your current drive.

DOS (PC only)

This is a gateway to the operating system. It can be used from the command line either alone or as a prefix to an operating system command.

To access the operating system environment, simply select the DOS command with the mouse or enter **`dos`** on the command line. To re-enter the XACT Development System, enter **`exit`** at the operating system command prompt.

If you want to execute a system command without leaving the Design Manager, enter it in the following manner.

`dos` *command*

This executes the command or program and returns you to XDM after the command or program finishes.

Edit

The Edit command brings up a text editor. To specify the text editing program that the Edit command uses, you must set an environment variable in your autoexec.bat or .cshrc file. The variable Editor should be set equal to the name of your text editing program as per the following:

```
SET EDITOR=EDIT           (DOS)
setenv EDITOR vi          (UNIX)
```

If this environment variable is not set and you select the Edit command, you will see an error message and return to XDM.

Browsing a file within the Design Manager can be slow on a PC and should only be used if you don’t have a text editor on your system. Use Edit instead if possible.

For example, if you are using a PC system and you want to invoke an external text editor with the push of a button, program one of the function keys. If you like the EDIT editor, for example, enter the following on the command line:

```
keydef f2 dos edit\
```

F2 is now programmed to invoke the EDIT editor.

The backslash (\) causes XDM to prompt you to finish the command when you press the defined key. You can use the backslash with any command that takes a variable argument at the end of its syntax. For example, if you define the F2 key as shown above and then press F2, you can answer the prompt with a file name as follows:

```
Cmd: dos edit myfile.txt
```

On a workstation, this option is not really necessary, because you can simply open another window in which to edit.

Execute

The execute command allows you to execute command files inside XDM. If you save a sequence of commands in a text file, you can execute them by first selecting Execute, then entering the command file name when prompted. You will return to XDM when the file has finished executing.

Help

XDM includes on-line help text for each menu, program or command, and program or command option. For example, you can display Help information about the XDM Verify menu, the VSM program located in the Verify menu, or the VSM -f option.

The Help command provides several methods for getting assistance in dealing with a particular topic. The normal method for using the help feature is to select the menu item in question and press the F1 function key. On workstations, click in the help window, and while the cursor is still in the help window press F1.

However, you can also get help by selecting the Help command from the Utilities menu, which prompts you for a help topic at the bottom of the screen:

```
Enter help subject:
```

An alternative method for displaying help text is to enter the following at the command line, replacing *topic* with whichever topic you need.

`help` *topic option*

For example, to display Help information about the -f option of the VSM command, enter the following at the command line, with no spaces between the program name and the option:

```
help vsm-f
```

Report

The Report command invokes the Version command, but instead of displaying the output on your screen, the output is redirected to a text file called version.rpt. The text file can then be read at any time using the Browse command.

ScanDisk

This command forces XDM to scan the hard disk drive to determine which supported software packages are installed on your system. While scanning, XDM displays the following message.

```
Checking disk for supported software...
```

This indicates that XDM is analyzing your system and setting up the contents of the DesignEntry, Translate, Fitter, and Verify menus, so they reflect the software that is available.

Then XDM displays a message similar to the one shown here.

```
Writing c:\xact\data\proglist.xdm...      (DOS)
Writing $XACT/data/proglist.xdm...        (UNIX)
```

And after a short while,

```
Writing c:\xact\data\proglist.xdm... done (DOS)
Writing $XACT/data/proglist.xdm... done   (UNIX)
```

This indicates that a new, updated proglist.xdm has been generated.

Note: Newly installed XDM-supported programs on your system may not appear in the XDM menus until the ScanDisk command is issued.

Version

Displays all supported programs currently installed on your system, showing the location and version numbers for each program. XDM may not be able to determine the version number of some programs, such as Xilinx-supported third-party programs.

The Profile Menu

This menu serves as a tool for customizing XDM. Using the commands in this menu, you can alter such characteristics as screen graphics, mouse button settings, PC mouse port connections (if you are using a serial mouse), and device type and speed. Changes that you make from the default profile will only be valid for the current session until you save them.

Note: To save a customized profile, including family and part settings, mouse button settings, PC mouse type, and so on, use the Save-profile command below. This allows you to use the same settings automatically each time you start up XDM.

Cursor

The Cursor command allows you to select an arrow, a bug, or a cross as the cursor.

Family

The Family command tells XDM the family of devices you are using. For designing with EPLDs, choose XC7200 or XC7300. XDM only displays valid menu items and command options for the selected family.

KeyCursor

When this command is enabled, the arrow keys move the cursor through pull-down menus, and pressing Enter executes the selected option (you must, however, select one of the menu commands with the mouse first).

When KeyCursor is off, commands must be entered through the keyboard or selected from pull-down menus with the mouse. You can also use the up and down arrow keys in the command line to scroll through previously entered commands.

Keydef

The Keydef command is used to program your system function keys. After selecting this command, XDM prompts you for a key name (e.g., F1, F2, etc.) and then a function. The function can be any XDM command (for example, BROWSE, DOS, etc.).

If you are using a PC and want to use the DOS EDIT editor, for example, enter the following on the XDM command line:

```
Cmd: keydef f2 dos edit\
```

F2 is now programmed to invoke EDIT.

The backslash (\) causes XDM to prompt you to finish the command when you press the defined key. You can use the backslash with any command that takes a variable argument at the end of its syntax. For example, if you define the F2 key as shown above and then press F2, you can answer the prompt with a file name as follows:

```
Cmd: dos edit uart_eqn.pld
```

The backslash feature is available for both PCs and workstations.

Menucolors

The Menucolors command allows you to change the color of items displayed in menus. Help can be used on the individual commands in this menu for more information on their functionality and usage.

Mouse

The mouse command sets the function of each mouse button. The available functions are Select, Done, and Menu.

If you are using a PC and the mouse driver loaded is not Microsoft-compatible, the mouse command also sets the connection port. If a Microsoft-compatible mouse driver is loaded, the connection port is automatically determined by XDM. This is indicated by the following being displayed on the screen:

```
Mouse: MS Mouse
```

If the driver loaded is not Microsoft-compatible, you must select the connection port through this command. In this case, the selected port name will be displayed on the screen (for example, “Mouse: COM1”).

Options

This command allows you to select default options for all the Xilinx software programs. For example, XEMAKE will use the part type and directory options you selected by reading xdm.pro. Once selected, they are valid for the current session. Use the SaveProfile command to save these options in the xdm.pro file.

Palette

You can choose different color palettes to customize your screen color by selecting any of the displayed palettes.

Part

The Part command allows you to select a default part type to use when translating a design.

Printer

The Printer command allows you to select a printer type for output files. The default type is POSTSCRIPT.

Readprofile

This command allows you to load the profile saved in the xdm.pro file. This command will first attempt to read a custom profile from the current directory. If one is not found it will load the default configuration profile in the \XACT\DATA or *$XACT*/data directory.

Saveprofile

This command allows you to save the current profile (which includes the family and part type, mouse mode, and so on) into an xdm.pro file in the current directory. Each time XDM is invoked it will try to read an xdm.pro file from the current directory. If one is not found it will load the default configuration profile in the \XACT\DATA or *$XACT*/data directory.

Settings

Selecting this command displays the current profile configuration of your Design Manager software.

Speed

The Speed command allows you to select a speed grade for the device specified with the Part command.

Report Formats

The XEPLD software creates a variety of report files in the design directory that provide you with information about the state of your design.

The *XEPLD Design Guide* explains how to use these reports to evaluate your design.

The report formats are described in the following sections.

Viewing Reports

Use the XDM **Utilities** → **Edit** command or the **Utilities** → **Browse** command to view report files.

The reports described in this chapter are:

- Resource report (*design_name*.res)
- Mapping report (*design_name*.map)
- Pinlist report (*design_name*.pin)
- Partitioner report (*design_name*.par)
- Logic Optimizer report (*design_name*.lgc)
- PLUSASM Assembler Log report (*pld_name*.lga)
- PAL Interconnect report (*design_name*.int)
- EQN file (*design_name*.eqn)

Other reports you should be aware of are the Error report (*design_name*.err), which records all diagnostic messages displayed on the screen during the fitting process, and the log file (*design_name*.log), which records diagnostic and informational messages.

The Resource Report

The Resource report (*design_name*.res), shown in Figure 3-1, is the first report you should examine to determine the results of the fitting process. This report lists the used and remaining numbers of Function Blocks, Macrocells, and pins of each type.

Logic Resources

The Logic Resources section shows the number of used and remaining Function Blocks and Macrocells.

Required Pin Resources

The Required column of the Pin Resources section shows the number of input, output (excluding global control/output), I/O, FastClk, FOE, and CE pins required by the design.

Used Pin Resources

The Used columns show how many of each type of physical device pin are occupied to satisfy the I/O signals in your design; note that output-only or input-only signals in your design may occupy I/O pins on the device.

Remaining Pin Resources

The Remaining columns show how many pins are still available on the device.

Fast Inputs and Outputs

The Fast Input/Fast Output section lists how many fast inputs and fast outputs the design requires and how many of each are available on the device.

```
XEPLD, Version 5.0                                          Xilinx Inc.
                                Resource Report
 Circuit name: UARTPALC
Target Device: XC7354-10PC68                  Integrated: 11-10-93,  1:37PM

LOGIC RESOURCES


                 Used       Remaining
Function Blocks     4         2
Macrocells         30        24

PIN RESOURCES:

Type    Required  --------Used--------------  --------Remaining---------
                  I  O  I/O Fclk Foe Cen Tot  I  O  I/O Fclk Foe Cen Tot
Inputs    3       3       0                3  5      34                39
Outputs  12          0    8    0   2   2  12     0   34    2   0   0  36
I/Os      0               0                0         34                34
Fclks     1                    1           1               2           2
Foes      0                        0       0                   0       0
Cens      0                            0   0                       0   0
        ---     --- --  ---  --- --- --- ---
         16       3  0    8    1   2   2  16

Note:The design requires 0 pins with Fast Input capability.
     This device has 11 pins with Fast Input capability.
     The design requires 0 pins with Fast Output capability.
     This device has 0 FO and 15 I/FO remaining from original 0 FO and 18 I/FO.

 End of Resource Report
```

Figure 3-1 Resource Report

The Mapping Report

The Mapping report (*design_name*.map), shown in Figure 3-2, lists the contents of all the macrocells and Function Blocks in the target device. For each Function Block in the device, a header area lists the total number of FastInputs, total inputs, shared product terms, and macrocells used by the design, and the total number available for the Function Block. Beneath each header is a list indicating how each macrocell in the Function Block is used. The list contains the following columns and sections.

For XC7272 designs only, the Function Block header also has a Carry value, which can be Low, Pterm, High, or Next FB. The Carry value indicates that a carry-in to the Function Block exists.

Function Name

This column shows the names of the signals in the design. For a behavioral design, the function name is the signal name.

For a schematic design, the function name is in the following format:

instance_name:*pin_name*

The instance name is the internal component name. The pin name is the name of the net that connects the I/O buffer and the pad symbol.

Macrocell Location

This column shows the Function Block and macrocell addresses where the component output pins or equation output signals are placed. The format of this column is: FB*function_block* – *macrocell*. For example, FB4-3 signifies Function Block 4, macrocell 3.

If you have specified low power on some macrocells, an L will appear on these macrocells in this report.

If a macrocell is using the macrocell carry from the previous macro-cell, a C will appear on the macrocell in this report.

Pkg Pin

This column shows the pin numbers of the device output or I/O pins that can be driven by the macrocells. A dot in this column signifies that the macrocell is buried and cannot drive a device pin.

Pin Use

This column shows how the corresponding device pins are used by the design (I, O, I/O, or dot). An “I” indicates that the pin is not used for macrocell output but is used by an input port in the design to receive some other type of signal.

Power Estimation (7300 Family Only)

At the end of this report is an estimate of the power consumption for low power and high power settings.

- The MCHP value is the number of used macrocells in used Function Blocks not designated as low power.
- The MCLP value is the number of used macrocells designated as low power plus the number of unused macrocells in used function blocks.
- Macrocells in unused Function Blocks do not consume power, therefore they are not counted in the power estimation calculation.

For more details about the power estimation calculation, see the Xilinx EPLD data sheets.

```
XEPLD, Version 5.0                                          Xilinx Inc.
                                 Mapping Report
 Circuit name: UARTPALC
Target Device: XC7354-10PC68                  Integrated: 11- 5-93, 12:23PM

Function Block 1              Fast inputs  -  0/12    Shared pterms -  0/0
                              Total inputs - 10/24    Macrocells    -  3/9

Function                                       Macrocell     Pkg      Pin
Name                                           Location      Pin      Use
--------                                       ---------     ---      ---
C2                                             FB1-1         4         .
START                                          FB1-2         12        .
PAR                                            FB1-3         13        .
.                                              FB1-4         15        .
.                                              FB1-5         17        .
.                                              FB1-6         19        .
.                                              FB1-7         21        .
.                                              FB1-8         22        .
.                                              FB1-9         23        .
...
Function Block 6              Fast inputs  -  0/3     Shared pterms -  1/12
                              UIM inputs   - 15/21    Macrocells    -  9/9

Function                                       Macrocell     Pkg      Pin
Name                                           Location      Pin      Use
--------                                       ---------     ---      ---
BITCLK                                         FB6-1         27        .
D4                                             FB6-2         28        .
D3                                             FB6-3         8         I
DOUT2                                          FB6-4         9         O
BYTECLK                                        FB6-5         10        .
D2                                             FB6-6         29        .
C0                                             FB6-7         6         .
C1                                             FB6-8         24        .
C5                                             FB6-9         26        .

Power estimation parameters: MCLP =   6, MCHP =  30.
MCLP represents macrocells in low power mode and
MCHP represents macrocells in high power mode.
Macrocells in unused function blocks are not considered.
MCLP and MCHP are used to estimate chip power consumption.
Consult the proper data sheets for this calculation.

 End of Mapping Report
```

Figure 3-2 Mapping Report (abbreviated)

The Pinlist Report

The Pinlist report (*design_name*.pin), shown in Figure 3-3, lists all pins on the target device, how they are used, and the names of their associated signals.

Pkg Pin

This column shows all the pin numbers of the target EPLD device, in numerical order, or in alphanumeric order in the case of PGA packages.

Pin Type

This column shows whether the pin is physically input-only (I), output-only (O), bidirectional (I/O), power (VCC, VSS), or special-purpose (MR, CLK, CEN, FOE).

Pin Use

This column shows how the pin is used in your design (I, O, I/O, and so on; see “Pin Use Legend”). For unused input and I/O pins, this column lists “tie” to remind you to tie the unused inputs to VCC or ground on your board. “NC” is shown for unused output-only pins indicating the pin should be left unconnected. If you select the “drive unused I/O pins” option of XEMAKE, FITEQN, or FITNET, an (O) appears in this column whenever an instance is mapped to the macrocell but the output is not used.

Pin Name

This column shows the name of the external signal mapped to the pin.

Pin Use Legend

This section lists the codes for the types of pins shown in the Pin Use column.

```
XEPLD, Version 5.0                                          Xilinx Inc.
                                Pin-List Report
 Circuit name: UARTPALC
Target Device: XC7354-10PC68                  Integrated: 11- 5-93, 12:23PM

Pkg  Pin   Pin    Pin
Pin  Type  Use    Name
---  ----  ---    ----
1    MR
2     I    tie    (unused)
3     I    tie    (unused)
4    I/O   tie    (unused)
5     I    tie    (unused)
6    I/O   tie    (unused)
7    VSS
8    CLK    I     X4CLK
9    CLK    O     DOUT2
10   CLK   tie    (unused)
...
58   I/O   tie    (unused)
59   VCC
60   CEN    O     DOUT1
61   CEN    O     DOUT0
62   FOE    O     FRAMING
63   VCC
64   FOE    O     PARITY
65    I     I     RD
66   I/O   tie    (unused)
67    I     I     CS
68    I    tie    (unused)

Pin Use Legend:

I     - input
O     - output
I/O   - input/output
I-L   - input uses latch
I-R   - input uses register
I/O-L - input/output uses latch
I/O-R - input/output uses register
NC    - not connected/not available
tie   - unused pin must be tied to VCC or GND
(O)   - unused pin attached to used macrocell

 End of Pin-List Report
```

Figure 3-3 Pin-List Report (abbreviated)

The Partitioner Report

The Partitioner report (*design_name*.par), shown in Figure 3-4 and Figure 3-8, shows the allocation of Function Block resources. This report helps you to understand how the resources available on the device were filled by the equations in your design. It can provide useful information for optimizing or modifying your design.

A separate partitioner report is created if you use XEMAKE, FITEQN, or FITNET. If you use PLUSASM, JED2PLD, ABL2PLD, or PALCONVT, a Partitioner report is included in the .log file.

The Partitioner report lists all partitions created to implement the design, and for each partition lists the outputs and Function Block resources. It also shows the input signals used by each partition and each output. Use this report when trying to increase the design density, or when modifying a design with a frozen pinout.

Summary

The summary contains information that applies to all partitions.

Part Name

This column lists partition names. For an individual PAL component report, names are the component name followed by a number.

For a report on a complete behavioral design, partition names are Function Block names. If the number of partitions in the design exceeds the number of Function Blocks in the device, the extra partitions will be named OVERFLOW_*n*, starting with OVERFLOW_0.

If you see extra partitions even though your design has fewer partitions than the device, it means some of the partitions in your design cannot use the available function blocks. For example, a partition that requires output pins cannot use a Function Block that has no pins.

Number of Outputs

This column shows the number of macrocells used in the partition.

Number of Input Lines Used

This column shows the combined number of pin and UIM inputs going into the partition. Fast Function Blocks allow up to 24 inputs. High Density Function Blocks allow up to 21 inputs.

Signal Inputs (Complete Design Only)

This column shows the number of inputs that would be necessary if no UIM ANDing were performed.

Number of Shared Pt

This column shows the combined number of shared product terms used in the partition. High Density Function Blocks have 12 shared product terms available. Fast Function Blocks have none.

O/IO Used (Complete Design Only)

This column shows output pins and I/O pins used by each partition.

O/IO Avail (Complete Design Only)

This column shows the total number of output pins and I/O pins available to each partition. The partitioner only counts usable outputs after considering pin assignment and control pin assignment. If you use a FastCLK signal as a control input, you cannot use the attached macrocell for an output pin. If you assign an input signal to an I/O pin, you cannot use the attached macrocell for an output pin.

Size Factor

This column estimates the Function Block resources used (from 0 to 9) based on the number of inputs, outputs, and shared product terms.

Inputs Used by Each Partition

This cross-reference table identifies the specific inputs feeding into each partition. Indexing numbers (top and bottom) for the inputs are listed at the end of the report. The Xs are inputs entering the function via an input line; the @s are ANDed UIM inputs.

Partition Listing

There is one partition listing for each partition in the design. Each listing contains information that applies to one partition only.

Signals Used

This list displays all the pin or UIM inputs to the partition.

Anded UIM Inputs Used (Complete Design Only)

This list displays all UIM equations that are inputs to the partition.

Inputs Used by Each Output Table

This cross reference table identifies the specific inputs used by each output macrocell in the equation file. The indexing numbers for inputs are defined in the input name list at the end of the report.

MC No

This column shows the macrocell number of the Function Block.

Output Name

This column identifies the output signal name mapped to the corresponding macrocell.

Pin Req (Complete Design Only)

This column indicates that the output requires no pin (-), an output pin (O), or an I/O pin (I/O).

Pin Avl (Complete Design Only)

This column indicates that the pin available to the output is no pin (-), an output pin (O), an I/O pin (I/O), a FastCLK pin (FCLK), or a Fast Output Enable pin (FOE).

Sh Pt

This column indicates the combined number of shared D1 and D2 product terms used by each output (whether or not any of the product terms are actually shared with any other output).

Input Listing

The names of all UIM inputs in the design appear in this list. At the end are all the UIM equations. The numbers correspond to the columns of the “Inputs Used” tables.

```
XEPLD, Version 5.0                                          Xilinx Inc.
                            Partitioning Report
 Circuit name: UARTPALC
Target Device: XC7354-10PC68                  Integrated: 11- 5-93, 12:23PM

Part        #  of    # of Input  Signal   # of        O/IO    O/IO   Size
Name        Outputs  Lines Used  Inputs   Shared Pt   Req     Avail  Factor
FB1           3         10         10        0         0/0     0/9     4
FB2           0          0          0        0         0/0     0/9     0
FB3           9         19         19        0         6/0     4/3     9
FB4           9         15         15        0         5/0     0/6     9
FB5           0          0          0        0         0/0     0/9     0
FB6           9         15         15        1         1/0     2/6     9
             ---                                       ---     ---    ---
             30                                       12/0     6/42   31

Part                            Inputs Used by Each Partition
Name
                              |----+----1----+----2----+----3
FB1                            X.......X.......XXXXXXXX..
FB2                            ..........................
FB3                            X.XXXXXXXXX.....XXXXXXXXX.
FB4                            ..XX....X...XXXXXXXXX.X.XX
FB5                            ..........................
FB6                            ..XX...XX.XXX...XXXXXXX.X.
                              |----+----1----+----2----+----3

+++++++++++++++++++++++++++++++++  FB1  +++++++++++++++++++++++++++++++++
                           Signals Used:
  1: SDIN              19: C2                22: C5
  9: BITCLK            20: C3                23: START
 17: C0                21: C4                24: PAR
 18: C1

MC Output         Pin  Pin  Sh
No Name           Req  Avl  Pt
                              |----+----1----+----2----+----3
1  C2             -    I/O  0  ................XXX...X...
2  START          -    I/O  0  X...............XXXXXXX...
3  PAR            -    I/O  0  X.......X.............XX..
4  Unused              I/O     ..........................
5  Unused              I/O     ..........................
6  Unused              I/O     ..........................
7  Unused              I/O     ..........................
8  Unused              I/O     ..........................
9  Unused              I/O     ..........................
```

Figure 3-4   The Partitioner Report (abbreviated)

```
+++++++++++++++++++++++++++++++++  FB6  +++++++++++++++++++++++++++++++++
                           Signals Used:
  3: RD                  12: D2                 20: C3
  4: CS                  13: D3                 21: C4
  8: READY               17: C0                 22: C5
  9: BITCLK              18: C1                 23: START
 11: D1                  19: C2                 25: BYTECLK

MC Output         Pin  Pin  Sh
No Name           Req  Avl  Pt
                              |----+----1----+----2----+----3
1  BITCLK         -    I/O  0  ................XX....X...
2  D4             -    I/O  0  ........X...X.............
3  D3             -    FCLK 0  ........X..X..............
4  DOUT2          O    FCLK 0  ..XX.......X............X.
5  BYTECLK        -    FCLK 0  .......X........XXXXXX....
6  D2             -    I/O  0  ........X.X...............
7  C0             -    I/O  0  ................X.....X...
8  C1             -    I/O  0  ................XX....X...
9  C5             -    I/O  1  ................XXXXXXX...
                              |----+----1----+----2----+----3

All outputs placed in a partition.

In   Input Name
No

  1   SDIN
  2   X4CLK
  3   RD
  4   CS
  5   FRAMING
  6   PARITY
  7   OVERUN
  8   READY
  9   BITCLK
 10   D0
...
 16   D6
 17   C0
...
 22   C5
 23   START
 24   PAR
 25   BYTECLK
 26   D7
```

Figure 3-4   The Partitioner Report (continued)

```
Partitioner Listing for File: rcvr Fri Oct  1 14:39:38 1993


                            PARTITIONING REPORT

Part        #  of    # of Input           # of                       Size
Name        Outputs  Lines Used           Shared Pt                  Factor
RCVR          8         17                   0                         8
             ---                                                      ---
             8                                                        8
Size factor may differ during integration because of external pin constraints!

Part                            Inputs Used by Each Partition
Name
                              |----+----1----+----2
RCVR                           XXXXXXXXXXXXXXXXX
                              |----+----1----+----2

+++++++++++++++++++++++++++++++++  RCVR  +++++++++++++++++++++++++++++++++
                           Signals Used:
  1: X4CLK              7: C5                  13: PARITY
  2: C0                 8: READ                14: OVERUN
  3: C1                 9: SDIN                15: READY
  4: C2                10: D0                  16: PAR
  5: C3                11: START               17: BITCLK
  6: C4                12: FRAMING

   Output                   Sh
   Name                     Pt
                              |----+----1----+----2
    START                    0  XXXXXXX.X.X......
    BITCLK                   0  XXX.......X......
    BYTECLK                  0  XXXXXXX.......X..
    PAR                      0  X.......X.X....XX
    FRAMING                  0  XXXXXXXXXX.X.....
    PARITY                   0  XXXXXXXX....X..X.
    OVERUN                   0  XXXXXXXX.....XX..
    READY                    0  XXXXXXXX...XXXX..
    Unused                      .................
                              |----+----1----+----2


All outputs placed in a partition.
```

Figure 3-5   Partitioner Report for a Single Component

```
In   Input Name
No

  1   X4CLK
  2   C0
  3   C1
  4   C2
  5   C3
  6   C4
  7   C5
  8   READ
  9   SDIN
 10   D0
 11   START
 12   FRAMING
 13   PARITY
 14   OVERUN
 15   READY
 16   PAR
 17   BITCLK
```

Figure 3-5   Partitioner Report for a Single Component (continued)

The Logic Optimizer Report

This report (*pld_name*.lgc), shown in Figure 3-8, which is also called the Collapse Module Report, shows you how logic was optimized for your design.

Logic optimization occurs when an input used by an equation (component output) is substituted by the logic function that generated that input. In the Logic Optimizer report, the output function which survives and absorbs another function into it is called a “fanout”. The logic function being absorbed is removed from the network if it is absorbed by all of the fanouts which use it.

In the report, “output” refers to an equation (or component) output within the design, not an EPLD output pin.

Summary

This section lists the following statistics:

- The number of FastCLKs, FOEs, and input registers used.
- The levels of combinational logic before and after optimization.

  The number of levels of logic is calculated as follows:

  - The number of macrocells between a registered macrocell and another registered macrocell, plus 1.
  - The number of macrocells between a input pad and a registered macrocell.
  - The number of macrocells between a registered macrocell and an output pad, plus 1.
- The number of functions optimized into subsequent functions.
- The number of output buffers that copied the signals of their input. This happens when the logic optimizer moves sequential logic forward into buffers.

  The logic optimizer moves forward any logic, whether combinational or sequential, that is buffered by a 3-state buffer, a clocked buffer, or a non-controlled buffer. However, logic that itself contains a 3-state or clock equation is not moved forward into a buffer that contains similar control equations.
- The number of outputs removed from the network.

Device Specific Optimization

This section lists the signals that were assigned to FastCLKs, FOEs, or input registers.

Outputs that Were Collapsed

This section contains two tables, which are transpositions of each other. The first shows which outputs have been optimized by absorbing the logic of one or more of their inputs; the second shows which functions have been collapsed into one or more of their fanouts.

Outputs Removed from the Network

If all the fanouts of a particular function optimize (absorb) the logic of the function, the function no longer needs to be implemented using a separate macrocell in the device, and it is removed from the network. This section is a list of the output functions that have been removed.

```
---------------------------------------------------------------------------
XEPLD, Version 5.0                                                Xilinx Inc.
                         Collapse Module Report
 Circuit name: HORNM4
Target Device: 73108144                       Integrated: 11- 4-93,  1:54PM

Summary:
^^^^^^^^
2 inputs were pad registered.
5 outputs were fast clocked.
2 outputs were fast output enabled.
There were 7 levels of combinational logic before collapsing,
and 2 levels after collapsing.
27 collapses of inputs into their fanouts were tried.
10 primary output buffers copied the signals  of their input.
15 outputs were removed from the network.


Device specific optimization:
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
2 inputs were pad registered. NOTE: that, these
inputs now preload high, which may affect the initial
value of any expression using one or more of these
inputs
5 outputs were fast clocked.
2 outputs were fast output enabled.
-----------------------------------------------------
Optimized
 logic
-----------------------------------------------------
X14                  pad registered using fclk:  CLK3
X13                  pad registered using fclk:  CLK2
U1[/Y22_0]:/Y22_0    fast clocked using fclk:    CLK
U1[/Y22_1]:/Y22_1    fast clocked using fclk:    CLK
U47:QC               fast clocked using fclk:    CLK
U47:QB               fast clocked using fclk:    CLK
U47:QA               fast clocked using fclk:    CLK
Y10                  fast out enabled using foe: T2
Y9                   fast out enabled using foe: T1
-----------------------------------------------------
```

Figure 3-6   Logic Optimizer Report

```
Outputs that were collapsed:
^^^^^^^^^^^^^^^^^^^^^^^^^^^^
27 outputs were optimized.  The following list
shows how each output was optimized.
-----------------------------------------------------
Original                       Optimizes the logic
 Output                             from output
-----------------------------------------------------
U48:Y                          U18:Y
U143:Y                         U48:Y
Y10                            U49:W
Y10                            U44:Y
U10:Y                          U2:Y
U10:Y                          Y5
U1[/Y22_0]:/Y22_0              U8:Y
U1[/Y22_0]:/Y22_0              ‘U1[W125PLD0]:Y22_0
U1[/Y22_1]:/Y22_1              U2:Y
U1[/Y22_1]:/Y22_1              ‘U1[W125PLD0]:Y22_1
Y1                             U1[PIN22]:PIN22
Y2                             Y1
Y3                             U2:Y
Y3                             U8:Y
Y3                             ‘U1[W125PLD0]:PIN18
Y4                             U2:Y
Y4                             U8:Y
Y4                             Y3
‘U1[W125PLD0]:PIN15            U2:Y
U7:Y                           U10:Y
U7:Y                           ‘U1[W125PLD0]:PIN15
U8:Y                           U7:Y
Y5                             U2:Y
Y5                             U8:Y
Y6                             Y5
Y7                             ‘U1[W125PLD0]:PIN15
U49:W                          U8:Y
-----------------------------------------------------
```

Figure 3-6   Logic Optimizer Report (continued)

```
The following list transposes the above list and
shows what outputs were used in optimizing some
other output.
-----------------------------------------------------
Original                       Optimized
 output                          into
-----------------------------------------------------
U48:Y                          U143:Y
U10:Y                          U7:Y
Y1                             Y2
Y3                             Y4
U2:Y                           U1[/Y22_1]:/Y22_1
U2:Y                           U10:Y
U2:Y                           ‘U1[W125PLD0]:PIN15
U2:Y                           Y5
U2:Y                           Y4
U2:Y                           Y3
‘U1[W125PLD0]:PIN15            U7:Y
‘U1[W125PLD0]:PIN15            Y7
U7:Y                           U8:Y
U8:Y                           U1[/Y22_0]:/Y22_0
U8:Y                           Y5
U8:Y                           Y4
U8:Y                           Y3
U8:Y                           U49:W
Y5                             Y6
Y5                             U10:Y
U49:W                          Y10
U18:Y                          U48:Y
U44:Y                          Y10
‘U1[W125PLD0]:Y22_0            U1[/Y22_0]:/Y22_0
‘U1[W125PLD0]:Y22_1            U1[/Y22_1]:/Y22_1
U1[PIN22]:PIN22                Y1
‘U1[W125PLD0]:PIN18            Y3
-----------------------------------------------------
```

Figure 3-6 Logic Optimizer Report (continued)

```
Outputs removed from the network:
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
15 output were removed from the network.
-----------------------------------------------------
Outputs removed from the network
-----------------------------------------------------
U18:Y
U48:Y
U44:Y
‘U1[W125PLD0]:Y22_0
‘U1[W125PLD0]:Y22_1
U1[PIN22]:PIN22
‘U1[W125PLD0]:PIN18
U143:Y
U144:Y
U145:Y
U10:Y
U7:Y
U2:Y
U49:W
‘U1[W125PLD0]:PIN15
-----------------------------------------------------


 End of Collapse Module Report
```

Figure 3-6 Logic Optimizer Report (continued)

The PLUSASM Assembler Log Report

This report (*pld_name*.lga), shown in Figure 3-8 and Figure 3-8, is generated when you fit a schematic design that contains PAL symbols. One PLUSASM assembler report is written for each PAL.

This report is written to a *pld_name*.lga file when you invoke the PLUSASM command or any command that calls PLUSASM (such as XEMAKE). In addition, commands that call FITEQN or FITNET include a PLUSASM log report in their .log files.

This report has two formats: one for the PLFB9 and PLFFB9 PALs (Figure 3-8), and one for the remaining PALs, including PL20V8, PL22V10, and the generic PALs (Figure 3-8).

Both formats show you a listing of your PLD equation file with each line numbered. Any syntax errors are indicated using an arrow marker on the line beneath your equation, along with a message. Additional errors or warnings appear at the end.

The PLFB9/PLFFB9 format begins with the equation file listing, which is followed by product term information.

The format for the other PALs begins with the partitioner log report and ends with a listing of the equation file.

Product Term Allocation (PLFB9/PLFFB9 Format)

This section shows how product term resources are allocated for each equation.

Pin

This column shows the pin numbers on the PLD symbol in a schematic) or an internally generated sequence number for a behavioral design.

Name

This column shows the names of all the signals in the equation file corresponding to the assigned pin numbers.

Type

This column shows how the PLD pins (signals) were used in the equation file.

Local P-terms Available

This column shows how many of the private product terms of each corresponding output macrocell are available to the logic function. When set, reset, clock, or output-enable control equations are used, the number of private product terms available for logic is reduced.

Local P-terms Used

This column shows the number of private product terms actually used by each output.

Shared D1 P-terms Used and Shared D2 P-terms Used

These columns show the amount of shared product term resources used by each output (regardless of whether any of the p-terms are actually shared by other outputs).

Total P-terms Used

This column is the sum of private and shared product terms used by each output.

Partitioner Log Report (Standard PAL Format)

This section shows how the PAL file was partitioned. It is an abbreviated form of the Partitioner report.

```
---------------------------------------------------------------------------
Start of PLUSASM ASSEMBLER LOG report

PlusAsm Listing for File: w1pld9.pld Wed Nov  3 13:32:21 1993

LINE # |----+----1----+----2----+----3----+----4----+----5----+----6----+---
     1
     2          TITLE      LIMIT CONDITIONS  CLFK NO RSTF NO SETF NO TRST NO
REGISTERED
     3
     4          AUTHOR     ISAAK V.
     5
     6          COMPANY    XILINX
     7
     8          DATE       4/25/93
     9
    10          PATTERN    FILE W1PLD9.PLD
    11
    12          CHIP       W1PLD9  PLPLD9
    13
    14
    15   ;1     2      3      4      5      6      7      8      9      10
    16
    17    X1    X2     X3     X4     X5     X6     X7     X8     X9     X10
    18
    19   ;11    12     13     14     15     16     17     18     19     20
    20
    21    X11   X12    X13    X14    X15    X16    X17    X18    X19    X20
    22
    23
    24   ;21    22     23     24     25     26     27     28     29     30
    25
    26    X21   Y1     Y2     Y3     Y4     Y5     Y6     Y7     Y8     Y9
    27
    28
    29
    30   ; 31    32
    31
    32     CLK   NC
...
    41    EQUATIONS
    42
    43    Y1 :=
X1*X2*X3*X4*X5*X6*X7*X8*X9*X10*X11*X12*X13*X14*X15*X16*X17*X18*X19*X20*/X21 +
    44
...
```

Figure 3-7   PLFB9 PLUSASM Assembler Log Report

```
   288                        X1*X2*X3*X4*X5*X6*X7*X8*X9*X10*/
X11*X12*X13*X14*X15*X16*X17*X18*X19*X20*X21 +
   289
   290                        X1*X2*X3*X4*X5*X6*X7*X8*X9*/
X10*X11*X12*X13*X14*X15*X16*X17*X18*X19*X20*X21 +
   291
   292                        /X9 + /X10+ /X11+ /X12
   293
   294
   295                        Y1.CLKF = CLK
   296                        Y2.CLKF = CLK
   297                        Y3.CLKF = CLK
   298                        Y4.CLKF = CLK
   299                        Y5.CLKF = CLK
   300                        Y6.CLKF = CLK
   301                        Y7.CLKF = CLK
   302                        Y8.CLKF = CLK
   303                        Y9.CLKF = CLK

No errors found in w1pld9.pld.

Pins and Product Term Allocation:

---------------------------------------------------------------------------
| Pin | Name       | Type      | Local    | Local | Shared | Shared | Total |
|     |            |           | Pterms   | Pterms| D1 Pt. | D2 Pt. | Pterms|
|     |            |           | Available| Used  | Used   | Used   | Used  |
---------------------------------------------------------------------------
    1   X1           input
    2   X2           input
...
   20   X20          input
   21   X21          input
   22   Y1           output         5        5       4        7        16
   23   Y2           output         5        5       4        7        16
...
   30   Y9           output         5        5       4        7        16
   31   CLK          fast clock


Input  Pins: 22
Output Pins:  9
```

Figure 3-7   PLFB9 PLUSASM Assembler Log Report (continued)

```
Shared Product Term Mapping

Y1
 Y2
  Y3
   Y4
    Y5
     Y6
      Y7
       Y8
        Y9
012345678                D1 Shared Product Terms
XXXXXXXXX  X1 * X2 * X3 * X4 * X5 * X6 * X7 * X8 * X9 * X10 * X11
           * X12 * X13 * X14 * X15 * X16 * X17 * X18 */X19 * X20
           * X21
...
XXXXXXXXX  X1 * X2 * X3 * X4 * X5 * X6 * X7 * X8 * X9 */X10 * X11
           * X12 * X13 * X14 * X15 * X16 * X17 * X18 * X19 * X20
           * X21
012345678                D2 Shared Product Terms
XXXXXXXXX  X1 * X2 * X3 * X4 * X5 * X6 * X7 * X8 * X9 * X10 * X11
           * X12 * X13 * X14 * X15 * X16 * X17 * X18 * X19 */X20
           * X21
...
XXXXXXXXX  X1 * X2 * X3 * X4 * X5 * X6 * X7 * X8 * X9 * X10 */X11
           * X12 * X13 * X14 * X15 * X16 * X17 * X18 * X19 * X20
           * X21

Shared D1 Product Terms:  4
Shared D2 Product Terms:  7
Overflowed Product Terms:  0
Size Factor:  9

End of PLUSASM ASSEMBLER LOG report
```

Figure 3-7   PLFB9 PLUSASM Assembler Log Report (continued)

```
Start of PARTITIONER LOG report
Splitting output 'Y' with too many product terms for one cell!
     16 shared pterms needed (after using 5 private) but only 12 available.

Partitioner Listing for File: w100pld Tue Nov 30 16:13:03 1993


                            PARTITIONING REPORT

Part        #  of    # of Input          # of                          Size
Name        Outputs  Lines Used          Shared Pt                     Factor
W100PLD       3         21                  8                            9
             ---                                                        ---
             3                                                          9

Part                                Inputs Used by Each Partition
Name
                              |----+----1----+----2----+----3
W100PLD                        XXXXXXXXXXXXXXXXXXXXX...
                              |----+----1----+----2----+----3

+++++++++++++++++++++++++++++++++  W100PLD  +++++++++++++++++++++++++++++++++
                           Signals Used:
  1: X1                 8: X7                    15: X15
  2: X12                9: X8                    16: X16
  3: X2                10: X9                    17: X17
  4: X3                11: X10                   18: X18
  5: X4                12: X11                   19: X19
  6: X5                13: X13                   20: X20
  7: X6                14: X14                   21: X21

   Output                   Sh
   Name                     Pt
                              |----+----1----+----2----+----3
    Y_0                      4  XXXXXXXXXXXXXXXXXXXXX...
    Y_1                      4  XXXXXXXXXXXXXXXXXXXXX...
    Y_2                      0  XXXXXXXXXXXXXXXXXXXXX...
    Unused                      ........................
    Unused                      ........................
    Unused                      ........................
    Unused                      ........................
    Unused                      ........................
    Unused                      ........................
                              |----+----1----+----2----+----3
```

Figure 3-8   Standard PLUSASM Assembler Log Report

```
All outputs placed in a partition.



In   Input Name
No

  1   X1
  2   X12
  3   X2
  4   X3
  5   X4
  6   X5
  7   X6
  8   X7
  9   X8
 10   X9
 11   X10
 12   X11
 13   X13
 14   X14
 15   X15
 16   X16
 17   X17
 18   X18
 19   X19
 20   X20
 21   X21
 22   Y_0
 23   Y_1
 24   Y_2



 End of Partitioning Report
---------------------------------------------------------------------------
End of PARTITIONER LOG report
```

Figure 3-8   Standard PLUSASM Assembler Log Report (cont.)

The PAL Interconnect Report

The PAL Interconnect report (*design_name*.int), shown in Figure 3-10, shows the signals used in each PAL file, ordered by pin number. The report is written to a file named *design_name*.int when you invoke the PALCONVT or FITEQN command to fit a behavioral design with multiple modules.

PAL Pin

This column displays the pin numbers in the PAL file pinlists.

Signal Name

This column displays the name of the signal routed to that pin.

PAL Use

Rows in this column have values of I, O, or O,FBK based on how the signal is used in equations.

Chip Use

Rows in this column have values of INPUTPIN, OUTPUTPIN, IOPIN, NODE, or FASTCLOCK based on how the software interprets the way the signal is used in the EPLD device.

Connectivity

This column contains values such as the following:

From PALA:3	Lists source if the signal is an input to the PAL.
To PALA:3, PALB:4	Lists input instances for an output generated by the PAL.
External Only	Indicates a signal connecting to or from a chip pin, but not between PALs.
Internal Only	Indicates a signal that connects only between equations in the same PAL.
CONFLICT with PALA:3	Indicates a signal used as an output more than once in the same design.

Unconnected Pins

This column lists the numbers of the pins that were not used in the design.

Summary

The Summary contains information about outputs, global control pins, and total number of pins of type input, output, and I/O in the following format:

```
Total number of Output Equations specified = 60
Total number of device pins used for Input = 45
Total number of device pins used for Output = 22
Total number of device pins used for I/O = 0
Total number of Global Control pins used = 3
```

```
XEPLD, Version 5.0                                          Xilinx Inc.
                        PAL INTERCONNECT REPORT
 Circuit name: UARTPALC
Target Device: 735468                         Integrated: 11- 5-93, 12:23PM

PAL File: shifter.pld +++++++++++++++++++++++++++++++++++++++++++++++++++++

PAL  Signal    PAL   CHIP
PIN  Name      Use   Use         Connectivity
 1   BITCLK    I     NODE        From rcvr.pld:16
 8   SDIN      I     INPUTPIN    External Only
12   D7        O     NODE        To datareg.pld:9
13   D6        O,FBK NODE        To datareg.pld:8
14   D5        O,FBK NODE        To datareg.pld:7
15   D4        O,FBK NODE        To datareg.pld:6
16   D3        O,FBK NODE        To datareg.pld:5
17   D2        O,FBK NODE        To datareg.pld:4
18   D1        O,FBK NODE        To datareg.pld:3
19   D0        O,FBK NODE        To rcvr.pld:10 datareg.pld:2

Unconnected pins:  2  3  4  5  6  7  9 10 11

PAL File: rcvr.pld ++++++++++++++++++++++++++++++++++++++++++++++++++++++++

PAL  Signal    PAL   CHIP
PIN  Name      Use   Use         Connectivity
 1   X4CLK     I     INPUTPIN    External Only
 2   C0        I     NODE        From cntr6.pld:19
 3   C1        I     NODE        From cntr6.pld:18
 4   C2        I     NODE        From cntr6.pld:17
 5   C3        I     NODE        From cntr6.pld:16
 6   C4        I     NODE        From cntr6.pld:15
 7   C5        I     NODE        From cntr6.pld:14
 8   READ      I     NODE        From cntr6.pld:13
 9   SDIN      I     INPUTPIN    External Only
10   D0        I     NODE        From shifter.pld:19
15   START     O,FBK NODE        To cntr6.pld:2
16   BITCLK    O,FBK NODE        To shifter.pld:1
17   BYTECLK   O     NODE        To datareg.pld:1
18   PAR       O,FBK NODE        Internal Only
19   FRAMING   O,FBK OUTPUTPIN   External Only
20   PARITY    O,FBK OUTPUTPIN   External Only
21   OVERUN    O,FBK OUTPUTPIN   External Only
22   READY     O,FBK OUTPUTPIN   External Only

Unconnected pins: 11 12 13 14
```

Figure 3-9 The PAL Interconnect Report

```
PAL File: datareg.pld +++++++++++++++++++++++++++++++++++++++++++++++++++++

PAL  Signal    PAL   CHIP
PIN  Name      Use   Use         Connectivity
 1   BYTECLK   I     NODE        From rcvr.pld:17
 2   D0        I     NODE        From shifter.pld:19
 3   D1        I     NODE        From shifter.pld:18
 4   D2        I     NODE        From shifter.pld:17
 5   D3        I     NODE        From shifter.pld:16
 6   D4        I     NODE        From shifter.pld:15
 7   D5        I     NODE        From shifter.pld:14
 8   D6        I     NODE        From shifter.pld:13
 9   D7        I     NODE        From shifter.pld:12
13   READ      I     NODE        From cntr6.pld:13
15   DOUT7     O     OUTPUTPIN   External Only
16   DOUT6     O     OUTPUTPIN   External Only
17   DOUT5     O     OUTPUTPIN   External Only
18   DOUT4     O     OUTPUTPIN   External Only
19   DOUT3     O     OUTPUTPIN   External Only
20   DOUT2     O     OUTPUTPIN   External Only
21   DOUT1     O     OUTPUTPIN   External Only
22   DOUT0     O     OUTPUTPIN   External Only

Unconnected pins: 10 11 12 14

PAL File: cntr6.pld +++++++++++++++++++++++++++++++++++++++++++++++++++++++

PAL  Signal    PAL   CHIP
PIN  Name      Use   Use         Connectivity
 1   X4CLK     I     INPUTPIN    External Only
 2   START     I     NODE        From rcvr.pld:15
 4   RD        I     INPUTPIN    External Only
 5   CS        I     INPUTPIN    External Only
13   READ      O     NODE        To rcvr.pld:8  datareg.pld:13
14   C5        O,FBK NODE        To rcvr.pld:7
15   C4        O,FBK NODE        To rcvr.pld:6
16   C3        O,FBK NODE        To rcvr.pld:5
17   C2        O,FBK NODE        To rcvr.pld:4
18   C1        O,FBK NODE        To rcvr.pld:3
19   C0        O,FBK NODE        To rcvr.pld:2

Unconnected pins:  3  6  7  8  9 10 11 12

Total number of Output Equations specified  = 31
Total number of device pins used for Input  = 4
Total number of device pins used for Output = 12
```

Figure 3-9   The PAL Interconnect Report (continued)

The EQN File

The Equation File (*design_name*.EQN), shown in Figure 3-10, is an expanded version of the .pld file that describes how your design was implemented on the EPLD device. The FITEQN and FITNET commands, and commands that call these commands, produce this report.

This file shows the following results of fitting the device:

- Where everything is placed in UIM functions
- FastCLK assignment
- FOE assignment
- Input register assignment
- Logic optimization results
- Logic minimizer results
- Equations assigned to a Fast Function Block with active-low polarity
- Export equations (Fast Function Blocks)
- Split equations

For information about how to use this report, see the *XEPLD Design Guide*.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This is the .eqn file produced by the partitioner.  It shows      ;
; how your equations were implemented in order to best utilize the  ;
; resources available on the chip.                                  ;
;                                                                   ;
; This design was compiled for the 735468                           ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

PATTERN UART.eqn

DATE  Thu Sep 23 15:31:23 1993

CHIP UART XEPLD

MINIMIZE OFF

PARTITION FB2_1 CONTROLLER:PIN15 CONTROLLER:PIN16 CONTROLLER:PIN18

PARTITION FB3_1 CONTROLLER:PIN17 ‘FREQ_DIVIDER[CB8RE]:Q4 DOUT1 DOUT0
                    ‘FREQ_DIVIDER[CB8RE]:Q5 READY PARITY OVERUN
                    FRAMING

PARTITION FB4_1 DOUT7 DOUT6 DOUT5 ‘FREQ_DIVIDER[CB8RE]:Q1
                    ‘FREQ_DIVIDER[CB8RE]:Q2 ‘FREQ_DIVIDER[CB8RE]:Q3
                    DOUT4 DOUT3 DOUT2

PARTITION FB6_1 DESERIALIZER:QA DESERIALIZER:QB DESERIALIZER:QC
                    DESERIALIZER:QH ‘FREQ_DIVIDER[CB8RE]:Q0
                    DESERIALIZER:QD DESERIALIZER:QE DESERIALIZER:QF
                    DESERIALIZER:QG

INPUTPIN SDIN RD CS
OUTPUTPIN ( FOE = $1N189 ) DOUT7 DOUT6 DOUT5 DOUT4 DOUT3 DOUT2 DOUT1
                   DOUT0
OUTPUTPIN  READY PARITY OVERUN FRAMING

NODE CONTROLLER:PIN15 CONTROLLER:PIN16 CONTROLLER:PIN17
                   CONTROLLER:PIN18 DESERIALIZER:QA DESERIALIZER:QB
                   DESERIALIZER:QC DESERIALIZER:QD DESERIALIZER:QE
                   DESERIALIZER:QF DESERIALIZER:QG DESERIALIZER:QH
                   ‘FREQ_DIVIDER[CB8RE]:Q0 ‘FREQ_DIVIDER[CB8RE]:Q1
                   ‘FREQ_DIVIDER[CB8RE]:Q2 ‘FREQ_DIVIDER[CB8RE]:Q3
                   ‘FREQ_DIVIDER[CB8RE]:Q4 ‘FREQ_DIVIDER[CB8RE]:Q5
FASTCLOCK X4CLK
FOEPIN $1N189
```

Figure 3-10   The EQN File

```
EQUATIONS
/CONTROLLER:PIN15 :=  ‘FREQ_DIVIDER[CB8RE]:Q0 */‘FREQ_DIVIDER[CB8RE]:Q1 */
‘FREQ_DIVIDER[CB8RE]:Q2
           * ‘FREQ_DIVIDER[CB8RE]:Q3 */‘FREQ_DIVIDER[CB8RE]:Q4
           * ‘FREQ_DIVIDER[CB8RE]:Q5
       +  SDIN */CONTROLLER:PIN15
   CONTROLLER:PIN15.CLKF =  X4CLK

/CONTROLLER:PIN16 := /CONTROLLER:PIN15
       +  ‘FREQ_DIVIDER[CB8RE]:Q1
       +  ‘FREQ_DIVIDER[CB8RE]:Q0
   CONTROLLER:PIN16.CLKF =  X4CLK

 CONTROLLER:PIN17 := /‘FREQ_DIVIDER[CB8RE]:Q0 * ‘FREQ_DIVIDER[CB8RE]:Q1 */
‘FREQ_DIVIDER[CB8RE]:Q2
           */‘FREQ_DIVIDER[CB8RE]:Q3 */‘FREQ_DIVIDER[CB8RE]:Q4
           * ‘FREQ_DIVIDER[CB8RE]:Q5 */READY
   CONTROLLER:PIN17.CLKF =  X4CLK

/CONTROLLER:PIN18 :=  SDIN * CONTROLLER:PIN18 * CONTROLLER:PIN16
       + /CONTROLLER:PIN18 */CONTROLLER:PIN16
       + /SDIN */CONTROLLER:PIN18
       + /CONTROLLER:PIN15
   CONTROLLER:PIN18.CLKF =  X4CLK

 FRAMING := /‘FREQ_DIVIDER[CB8RE]:Q0 */‘FREQ_DIVIDER[CB8RE]:Q1 */
‘FREQ_DIVIDER[CB8RE]:Q2
           * ‘FREQ_DIVIDER[CB8RE]:Q3 */‘FREQ_DIVIDER[CB8RE]:Q4
           * ‘FREQ_DIVIDER[CB8RE]:Q5 */DESERIALIZER:QA
       + /SDIN */‘FREQ_DIVIDER[CB8RE]:Q0 */‘FREQ_DIVIDER[CB8RE]:Q1
           */‘FREQ_DIVIDER[CB8RE]:Q2 * ‘FREQ_DIVIDER[CB8RE]:Q3
           */‘FREQ_DIVIDER[CB8RE]:Q4 * ‘FREQ_DIVIDER[CB8RE]:Q5
       +  CS * FRAMING
       +  RD * FRAMING
   FRAMING.CLKF =  X4CLK

 PARITY := /‘FREQ_DIVIDER[CB8RE]:Q0 * ‘FREQ_DIVIDER[CB8RE]:Q1 */
‘FREQ_DIVIDER[CB8RE]:Q2
           */‘FREQ_DIVIDER[CB8RE]:Q3 */‘FREQ_DIVIDER[CB8RE]:Q4
           * ‘FREQ_DIVIDER[CB8RE]:Q5 * CONTROLLER:PIN18
       +  CS * PARITY
       +  RD * PARITY
   PARITY.CLKF =  X4CLK

 OVERUN := /‘FREQ_DIVIDER[CB8RE]:Q0 * ‘FREQ_DIVIDER[CB8RE]:Q1 */
...
```

Figure 3-10   The EQN File (continued)

PLUSASM Command Reference

Introduction

This chapter defines the structure and syntax of the PLUSASM Boolean equation language. PLUSASM is the fundamental language used for expressing behavioral designs targeted for Xilinx EPLDs. You can use PLUSASM to define custom logic functions representing either an entire EPLD design, or just part of a design (such as a PLD equation file in a schematic). All standard components in the XEPLD schematic symbol library are defined by PLUSASM equation files.

PLUSASM is compatible with the PALASM equation language, except where noted. PALASM is commonly used for creating conventional PLD products, and most PALASM Boolean equation files can be read and processed directly by the XEPLD PLUSASM Assembler.

PLUSASM Overview

Using a text editor, you can create a complete PLUSASM design within a single Top-Level File, which contains both design control information and behavioral equations. However, it is often convenient to develop your design as separate Include Files and link them together within a Top-Level File to form a complete design.

If you are converting existing PAL designs, the XEPLD PALCONVT command automatically creates a Top-Level File which contains your PAL equation files as Include Files.

PLUSASM File Structure

The basic structure of PLUSASM files is illustrated in Figure 4-1. This figure shows a hierarchical design composed of a Top-Level File containing both embedded equations and three Include Files referenced by INCLUDE_EQN statements.

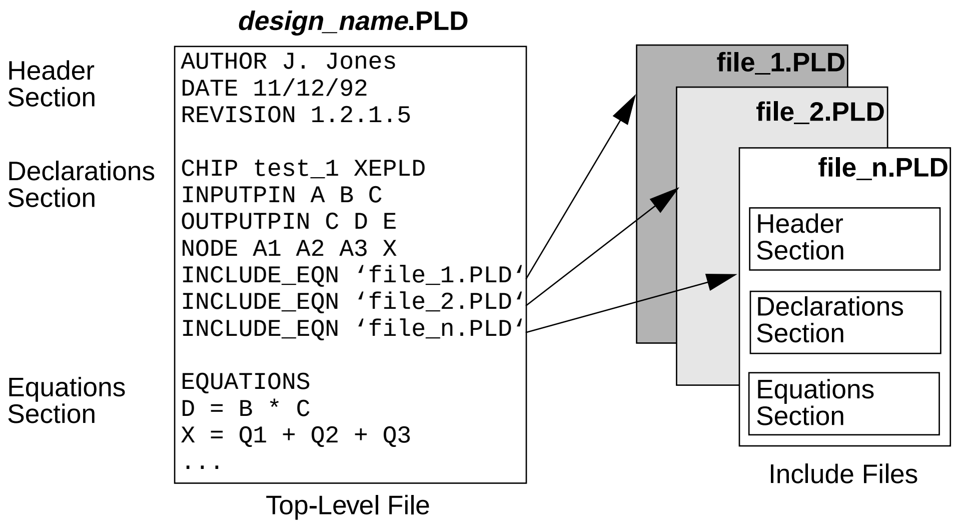


Figure 4-1 PLUSASM File Structure

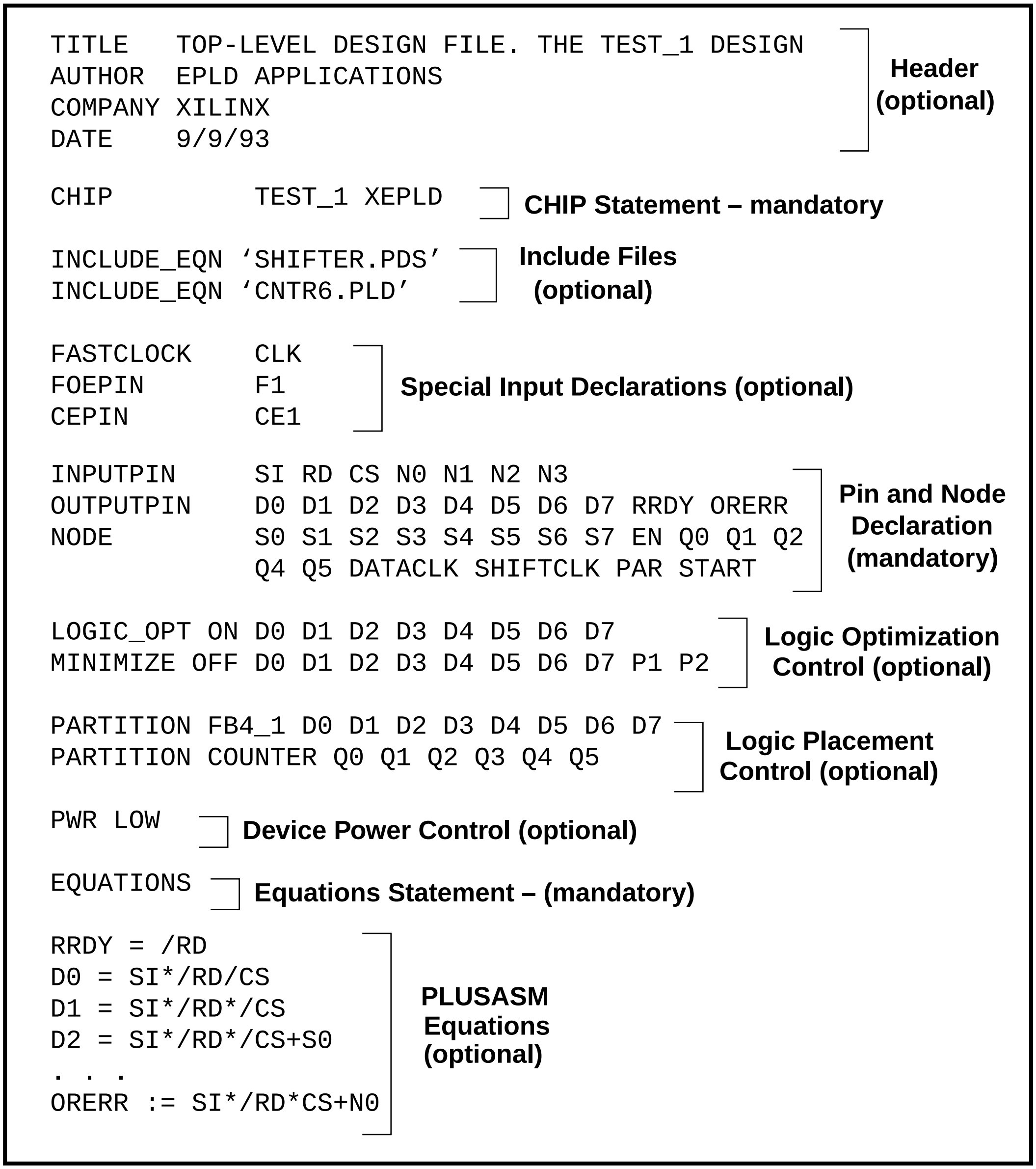
The file structure is identical for both the Top-Level File and the Include Files. However, the Include Files have a different CHIP statement syntax and they can contain only a limited subset of declaration statements.

These files have three key sections:

- The Header Section.
- The Declarations Section.
- The Equations Section.

The commands and keywords used in each section are described in the subsequent sections of this chapter.

A typical PLUSASM design file is illustrated in Figure 4-2.



```
TITLE   TOP-LEVEL DESIGN FILE. THE TEST_1 DESIGN
AUTHOR  EPLD APPLICATIONS
COMPANY XILINX
DATE    9/9/93

CHIP          TEST_1 XEPLD

INCLUDE_EQN ‘SHIFTER.PDS’
INCLUDE_EQN ‘CNTR6.PLD’

FASTCLOCK     CLK
FOEPIN        F1
CEPIN         CE1

INPUTPIN      SI RD CS N0 N1 N2 N3
OUTPUTPIN     D0 D1 D2 D3 D4 D5 D6 D7 RRDY ORERR
NODE          S0 S1 S2 S3 S4 S5 S6 S7 EN Q0 Q1 Q2
              Q4 Q5 DATACLK SHIFTCLK PAR START

LOGIC_OPT ON D0 D1 D2 D3 D4 D5 D6 D7
MINIMIZE OFF D0 D1 D2 D3 D4 D5 D6 D7 P1 P2

PARTITION FB4_1 D0 D1 D2 D3 D4 D5 D6 D7
PARTITION COUNTER Q0 Q1 Q2 Q3 Q4 Q5

PWR LOW

EQUATIONS

RRDY = /RD
D0 = SI*/RD/CS
D1 = SI*/RD*/CS
D2 = SI*/RD*/CS+S0
. . .
ORERR := SI*/RD*CS+N0
```

Header (optional)
CHIP Statement – mandatory
Include Files (optional)
Special Input Declarations (optional)
Pin and Node Declaration (mandatory)
Logic Optimization Control (optional)
Logic Placement Control (optional)
Device Power Control (optional)
Equations Statement – (mandatory)
PLUSASM Equations (optional)

Figure 4-2   A Typical PLUSAM Design File

The Header Section

Header statements are used to annotate your design for tracking and documentation purposes; they are ignored by XEPLD and have no effect on your design. All header statements are optional, they must precede the declarations section, and they may appear in any order. These statements are applicable to behavioral designs, PLD definitions in a schematic design, and Include Files. Each statement may contain up to a full line of alphanumeric text.

Header Statements

AUTHOR ***text***
COMPANY ***text***
DATE ***text***
PATTERN ***text***
REVISION ***text***
TITLE ***text***

```
TITLE design1 Hyperlink Controller
AUTHOR B. Wilson
COMPANY Little Blue
PATTERN test chip for hyperlink
REVISION 1.0.2.5a.7.21
DATE 7/8/92

CHIP design1 XEPLD; The Declarations section starts here
...
```

Figure 4-3 Header Statement Syntax Example

The Declarations Section

Declaration statements specify device I/O pins and affect how your behavioral equations are mapped into a specific device. The first statement (after the Header Section) must be the CHIP statement immediately followed by the pinlist. All other declaration statements may be used in any order.

Declaration Statement	**TOP**	**INC**	**Function Overview**
CEPIN	X		Specifies the global Clock Enable input pins.
CHIP	X	X	Specifies the file type, file name, and pin list.
FASTCLOCK	X	X	Specifies the global FastCLK inputs.
FASTINPUT		X	Declares the signals connected to the FastInput pins.
FOEPIN	X		Specifies global FAST Output Enable input pins.
INCLUDE_EQN	X		Specifies names of Include Files.
INPUTPIN	X		Specifies device input pins.
IOPIN	X		Specifies device I/O pins.
LOGIC_OPT	X		Controls logic optimization.
MINIMIZE	X	X	Controls the use of logic minimization.
MRINPUT	X		Specifies the Master Reset input.
NODE	X	X	Specifies nodes in the design.
OPTIONS	X		Controls the automatic use of device resources.
OUTPUTPIN	X		Specifies device output pins.
PARTITION	X	X	Specifies physical locations for groups of equations.
PWR	X		Controls the device power usage.
STRING	X	X	Specifies a global text string substitution.

TOP = Used in Top-Level Files

INC = Used in Include Files and PLD Files in Schematics

CEPIN

[CEPIN /*ce_name*... [PIN *pin_number* ...]]

Use the CEPIN statement in Top-Level Files only.

The CEPIN statement declares the signals received via the global Clock Enable (CE) pins of the XC7000 device. You can use signals defined as CEPINs only in the (CE = *ce_name)* option of the INPUTPIN and IOPIN statements. Signals declared as CEPIN must be active low.

You can also control global clock enable using on-chip logic. To do this, specify an output equation for a signal named as a CEPIN. The output function will then drive the device CE pin which in turn controls selected input pad registers

Note: Do not declare a CEPIN signal again as an OUTPUTPIN or IOPIN statement.

```
INPUTPIN (RCLK=CLK CE=ENAB1) X
CEPIN /ENAB1
FASTCLOCK CLK
...
EQUATIONS
ENAB1 = READ_EN * SELECT_1; controls CEPIN from on-chip
```

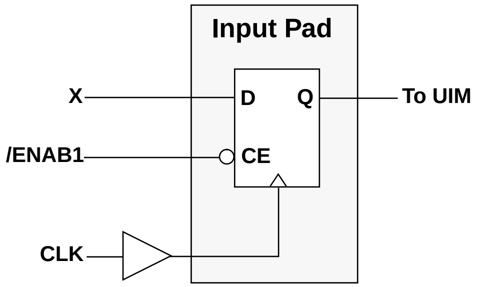


Figure 4-4 Using the CEPIN Statement

Note: The CEPIN statement must always be accompanied by a FASTCLOCK declaration.

CHIP

```
CHIP file_name device_name signal_list
```

Use the CHIP statement in Top-Level Files, Include Files, and PLD files.

The CHIP statement identifies an equation file with a unique name and target device type. It also declares all signals used in an Include File or PLD equation file. It is mandatory in all equation files.

file_name

Use *file_name* to specify the equation file name (without extension).

For Top-Level Files, *file_name* is the name of your design.

For schematic-based designs, *file_name* is the value of the PLD attribute placed on a PLD or custom symbol, or is the actual name of a user-defined primitive symbol.

device_name

Use *device_name* to define the target PLD type, which can be any name. However, some names have special meaning. The software recognizes the following:

- Any valid PLD name in the schematic library: PL20V8, PL22V10, PLFB9, and PLFFB9.
- The keyword XEPLD (used to specify the Top-Level file of behavioral designs only).
- The keyword COMPONENT (used to specify user-defined primitive symbols).
- Any valid device name: XC7236, XC7236A, XC7272, XC7272A, XC7318, XC7336, XC7354, XC7372, XC73108.

The following aliases are accepted for the PL20V8:

```
20V8, G20V8, GAL20V8, P20V8, P20V8R, PAL20V8.
```

The following aliases are accepted for the PL22V10:

```
22V10, G22V10, GAL22V10, P22V10, PAL22V10.
```

All other names are unrecognized and are treated as generic equation files.

If you specify any PLD type other than PLFB9, PLFFB9, 22V10 or 20V8 (or their aliases) the software will assume the PLD is a generic device. The equation files for generic PLDs must explicitly define all functionality; implied features are not recognized.

Note: If you are creating original equation files to be included in a behavioral or schematic design which is not targeted to a specific PLD type, you can use the name “GENERIC” as the PLD type.

signal_list

Use *signal_list* to declare all signal names appearing in the Include Files or in PLD files used in schematics. Each signal name must be separated by commas, tabs, spaces, or carriage returns. To indicate that a signal is active-low, precede the name with a slash (/).

For PLD components in a schematic, the signal list order corresponds to the order of pins on the PLD symbol, starting with the signal name corresponding to pin 1 on the component symbol. If a PLD pin position is not used, it must be labeled NC (No Connection) as a placeholder. GND and VCC are reserved words treated the same as NC, providing compatibility with equation files prepared for pin-compatible industry-standard PLDs. Refer to the pin descriptions in the PLD library data sheets for allowable positions of inputs, outputs, and I/Os in the signal list.

```
CHIP design1 22V10
CLK S1 /S2 NC NC Q0 Q1 Q2 Q3 Q4 Q5 GND
Q6 Q7 X1 X2 /X3 X4 X5 /X6 X7 N1 N2 VCC
; Used for PLD and Include files
```

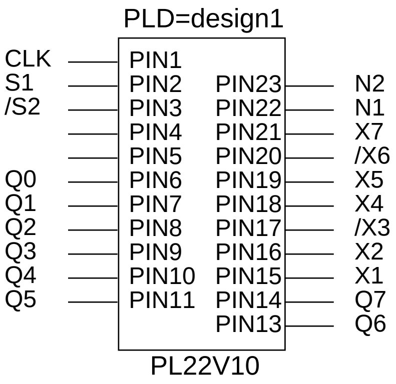


Figure 4-5 Using the CHIP Statement

FASTCLOCK

```
[FASTCLOCK fastclock_name ...
  [PIN pin_number ...]]
```

Use the FASTCLOCK statement in Top-Level Files, Include Files, and PLD files.

The FASTCLOCK statement declares the signals that are received via the global FastCLK pins of the XC7000 device. FastCLK signals are used for FB and FFB macrocell register clocks, input pad register clocks, and input pad latch enable inputs.

The first *fastclock_name* appearing in the FASTCLOCK statement of a behavioral design is used as the default clock source for any registered equations for which no *signal_name*.CLKF statement has been specified. The default clock pin for PLD components used in schematics is pin 1 unless otherwise specified in the components' data sheets.

Within the EQUATIONS section of any equation file, the *fastclock_name* can only be used in the *signal_name*.CLKF = *fastclock_name* equation.

You must use the FASTCLOCK statement to declare the signals used as clocks for input-pin registers and latches. For INPUTPIN and IOPIN statements, FASTCLOCK signals can be applied as follows:

- RCLK = *fastclock_name* (register clock specification). For example:

  ```
  FASTCLOCK clk_1
  INPUTPIN (RCLK=clk_1) sig_1 sig_2 sig_5
  ```

- LE = *fastclock_name* (latch enable specification). For example:

  ```
  FASTCLOCK ck_inp
  IOPIN (LE=ck_inp) Q0 Q1 Q2 Q3
  ```

The FASTCLOCK statement may also be used in a PLD component equation file (except PLFB9 or PLFFB9) to specify a PLD pin which can only be driven by a FastCLK signal. This informs the PLUSASM assembler that a Function Block input channel and clock p-term need to be allocated for that signal.

You can also generate a global FastCLK signal using on-chip logic. To do this, specify an output equation for a signal defined as a FASTCLOCK. The output function will then drive the device FastCLK pin, which in turn controls selected registers in the design.

Note: Do not declare a FASTCLOCK signal name again in an OUTPUTPIN or IOPIN statement.

Note: You may have as many FASTCLOCK statements in your design as there are FastCLK signals available in the specified XC7000 device.

```
INPUTPIN (RCLK=CLK1) X
OUTPUTPIN REG2
FASTCLOCK CLK1
...
EQUATIONS
REG2 := X
REG2.CLKF = CLK1
...
CLK1 = SYSCLK + INHIBIT_1; To control FATCLOCK on-chip
```

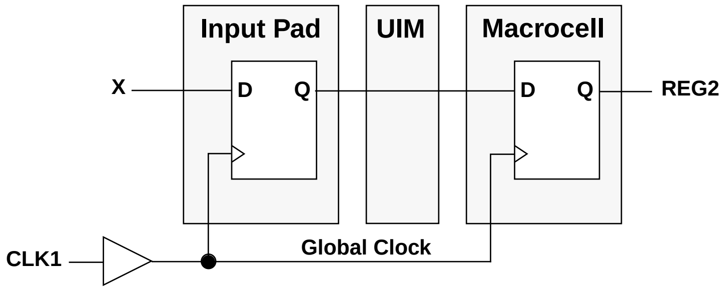


Figure 4-6 Using the FASTCLOCK Statement

Note: The first signal appearing in the FASTCLOCK statement is the default clock for all registered equations in the Top-Level File. The first signal appearing in the pinlist of an include file is the default clock for all registered equations in the Include File. Default clocks are overridden by the .CLKF control equation.

FASTINPUT

[FASTINPUT *input_name*...]

Use the FASTINPUT statement in Include Files and PLD files only.

The FASTINPUT statement declares the signals that are connected to the FastInput pins of the EPLD device. It is used only in equation files for PLFFB9 components in XC7300-series schematic designs. Declaring an input signal as a FASTINPUT means that the signal will be taken through the FastInput path, bypassing the UIM.

If both the FastInput path and the UIM input path from the same device pin are used, two pins of the PLFFB9 must be specified, one of which is named in a FASTINPUT statement.

Signals declared as FASTINPUT must also be listed in the equation file pin list and the corresponding PLD component pins must be connected directly to IBUF symbols in the schematic.

```
CHIP PLD3 PLFFB9
  X Y Z1
FASTINPUT X
...
EQUATIONS
Z1 = X + Y
```

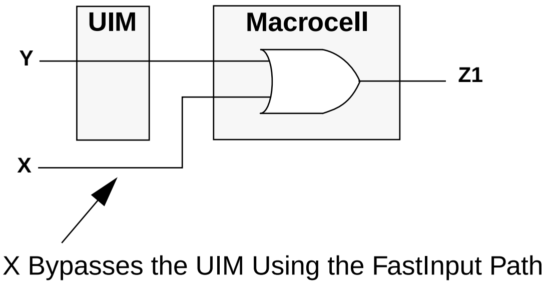


Figure 4-7 Using the FASTINPUT Statement

FOEPIN

[FOEPIN *foe_name*... [PIN *pin_number* ...]]

Use the FOEPIN statement in Top-Level Files only.

The FOEPIN statement declares the signals received via the global FOE pins of the EPLD device. You can use signals defined as FOEPINs only in the FOE = *foe_name* pin option of the OUTPUTPIN and IOPIN statements. FOEPIN is not used with the XC7272.

The Fast Output Enable (FOE) pins of the device allow the 3-state outputs to be quickly enabled or disabled by external signals. Because the FOE signals use dedicated signal wiring on the device, they are faster than the p-term tristate enable signals (.TRST equations) which use macrocell resources. Using the FOE pins for 3-state output control can also reduce device resource requirements because no macrocell resources are required.

To change a 3-state output enable from a p-term enable (expressed as .TRST) to an FOE enable (controlled by an external pin), do the following:

1. Add the "FOEPIN *signal*" statement to the Top-Level File.
2. Add the "FOE = *signal_name*" declaration to the OUTPUTPIN statement.
3. Delete the .TRST equations.

For example, using the .TRST equation for 3-state control:

```
OUTPUTPIN TEST
TEST = X*Y
TEST.TRST = OUT1
```

The same function, using the FOEPIN declaration:

```
OUTPUTPIN (FOE=OUT1) TEST
FOEPIN OUT1
TEST = X*Y
```

Note: You cannot use the FOEPIN signal as a logic input; FOEPIN affects PIN feedback but not UIM feedback. Only output enable signals that can be controlled by a single active-high pin may be assigned to an FOE net. Active-low output enable signals must come from a product term (macrocell).

You can also control Fast Output Enable using on-chip logic. To do this, specify an output function for a signal named as an FOEPIN. The output function will then drive the device FOE pin, which in turn controls selected 3-state output drivers.

```
FOEPIN OE
IOPIN (FOE=OE) IO1
...
EQUATIONS
IO1 = A + B
OE = /READ * /CHIP_SEL; To control FOE on-chip
```

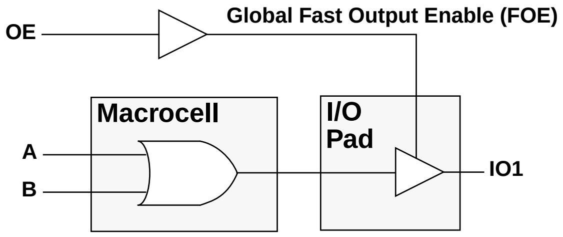


Figure 4-8 Using the FOEPIN Statement

INCLUDE_EQN

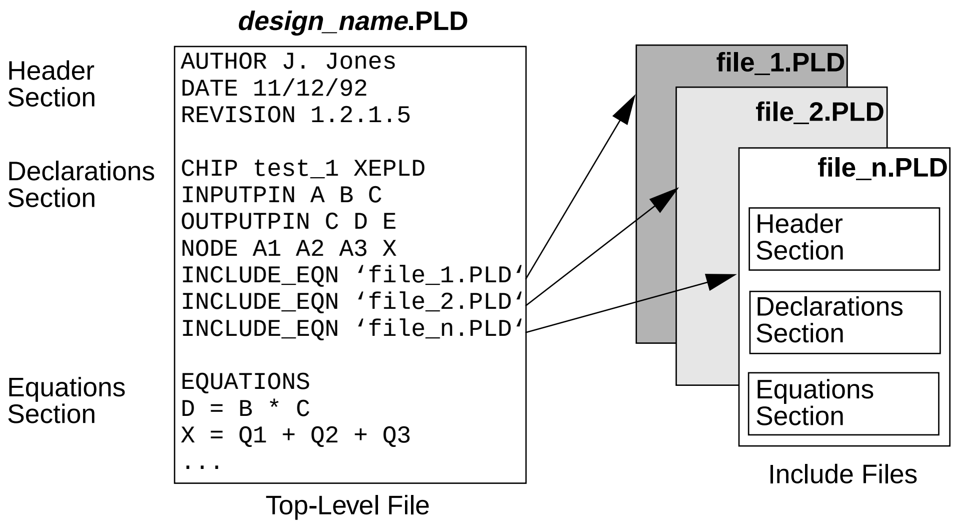
[INCLUDE_EQN '*file_name.extension*']

Use the INCLUDE_EQN statement in Top-Level Files only.

The INCLUDE_EQN statement is used to link external equation files (called Include Files) to the Top-Level File. Signal names used in Include Files are global; a signal name used in both a Top-Level File and its Included Files is interpreted as the same signal.

Include Files must have a .PLD or .PDS file name extension.

The PLUSASM Quick Reference section, at the end of this chapter, shows the PLUSASM commands that can be used within Include Files.



```
AUTHOR J. Jones
DATE 11/12/92
REVISION 1.2.1.5

CHIP test_1 XEPLD
INPUTPIN A B C
OUTPUTPIN C D E
NODE A1 A2 A3 X
INCLUDE_EQN ‘file_1.PLD‘
INCLUDE_EQN ‘file_2.PLD‘
INCLUDE_EQN ‘file_n.PLD‘

EQUATIONS
D = B * C
X = Q1 + Q2 + Q3
...
```

Figure 4-9 Using the INCLUDE_EQN Statement

INPUTPIN

```
[INPUTPIN [({RCLK = fastclock_name [CE = ce_name]
| LE = fastclock_name | FI})] [/] signal_name...
[PIN pin_number ...]]
```

Use the INPUTPIN statement in Top-Level Files only.

The INPUTPIN statement declares the signals received via EPLD device pins. A physical input-only or I/O pin is allocated to receive each signal. A design may contain several INPUTPIN statements specifying various combinations of pin options, including the following:

RCLK = *fastclock_name*

Each input is registered and clocked by *fastclock_name*.

CE = *ce_name*

Input register loading is enabled by *ce_name* which is an active-low enable (otherwise input registers are always enabled if CE is not specified). CE can only be used in XC7300-series designs and only if RCLK is specified.

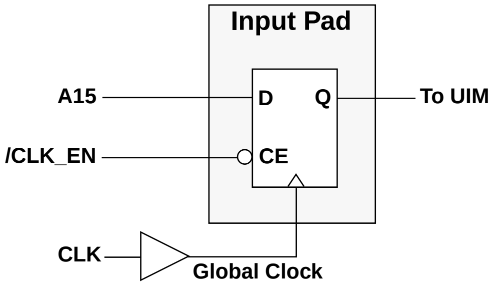
LE = *fastclock_name*

Each input is latched and the latch is made transparent while *fastclock_name* is High. LE is mutually exclusive to RCLK and CE.

FI

Each input is placed on a FastInput pin, and instances of each signal name refer by default to the FastInput path. (This applies only to XC7300-series designs.)

```
FASTCLOCK CLK
CEPIN /CLK_EN
INPUTPIN (RCLK=CLK CE=CLK_EN) A15
```



```
INPUTPIN (FI) MC_EN
INPUTPIN (LE=AS) A15
```

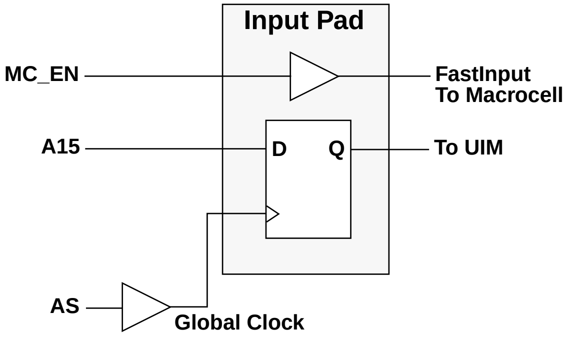


Figure 4-10   Using the INPUTPIN Statement

IOPIN

```
[IOPIN [([RCLK = fastclock_name [CE = ce_name] |
LE = fastclock_name] [PINFBK]
[FOE = foe_name] [NODETRST])] [/] signal_name...
[PIN pin_number ...]]
```

Use the IOPIN statement in Top-Level Files only.

The IOPIN statement declares the signals that drive and can be received from EPLD device I/O pins. A device I/O pin is allocated to each signal specified in an IOPIN statement.

Instances of I/O signal names can take their inputs from either the I/O pin or from the macrocell feedback. By default, feedback is taken internally from the macrocell. Applying the .PIN extension to the signal indicates that the instance uses input from the I/O pin. Use the PINFBK option to specify that inputs always come from the device pin instead of the internal macrocell feedback.

A design may contain several IOPIN statements specifying various combinations of pin options including the following:

RCLK = *fastclock_name*

Each input is registered and clocked by *fastclock_name*.

CE = *ce_name*

Input register loading is enabled by *ce_name* which is an active-low enable (otherwise input registers are always enabled if CE is not specified). CE can only be used in XC7300-series designs and only if RCLK is specified.

LE = *fastclock_name*

Each input is latched and the latch is made transparent while *fastclock_name* is High. LE is mutually exclusive to RCLK and CE.

PINFBK

Instances of each signal name, when used as inputs, refer to the I/O pin input rather than the internal macrocell feedback.

FOE = *foe_name*

Each 3-state output is enabled by *foe_name*. You can use this option in conjunction with *signal_name*.TRST equations for the same outputs. (FOE does not apply to XC7272 designs.)

NODETRST

Any *signal_name*.TRST equations specified for these outputs affect the macrocell feedbacks as well as the device pin drivers. (This is the assumed behavior of XC7272 designs, in which 3-state control always affects both pin drivers and the macrocell feedbacks.)

```
CHIP design1 XEPLD
FASTCLOCK ck1
CEPIN en2
FOEPIN f1
IOPIN (RCLK=ck1 CE=en2 FOE=f1 PINFBK) sig_A sig_B sig_C
IOPIN X1 X2 X3
EQUATIONS
```

```
IOPIN (PINFBK) CEO
EQUATIONS
CEO = A + B
CEO.TRST = OE
```

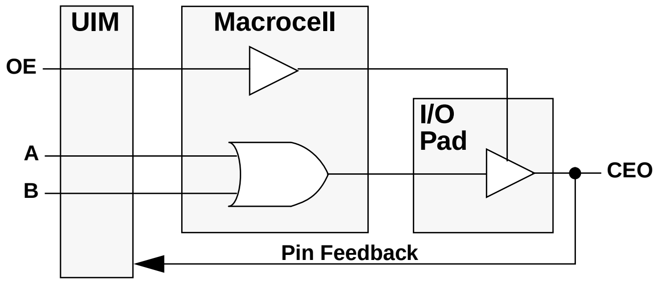


Figure 4-11 Using the IOPIN Statement with Pin Feedback

```
IOPIN (LE=AS) CNTR0
EQUATIONS
CNTR0 = A + B
```

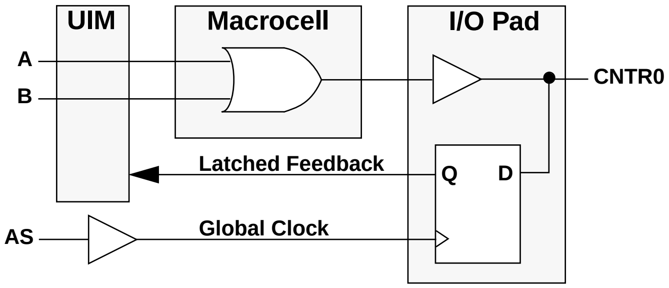


Figure 4-12   Using IOPIN with Latch Enable

LOGIC_OPT

```
[LOGIC_OPT {OFF | ON} [signal_name…]]
```

Use the LOGIC_OPT statement in Top-Level Files only.

Use the LOGIC_OPT statement to inhibit the automatic logic collapsing function of the XEPLD software. If you omit the output list, the statement applies to all signals. If you include an output list, the statement applies to the named signals and the opposite setting applies to all other signals. Only one LOGIC_OPT statement is allowed per design. LOGIC_OPT ON is the default.

Logic collapsing produces a more compact and usually faster design requiring fewer device resources. You will need to verify circuit timing based on the collapsed version of your circuit as shown in the *design_name*.eqn report.

Note: The logic collapsing algorithm attempts to reduce the number of macrocells. For example:

```
A := B * C ; requires 1 macrocell
B = D + E ; requires 1 macrocell
```

These separate equations require two macrocells. However, with LOGIC_OPT ON, they collapse to a single output equation as follows:

```
A := C*D + C*E ; requires 1 macrocell
```

Specifying LOGIC_OPT OFF for a node ensures that a macrocell output will be used to generate that node. LOGIC_OPT OFF prevents the node from being collapsed forward into any succeeding equations. LOGIC_OPT OFF also prevents a single p-term from being completely optimized into the UIM; a macrocell will still be used to produce the function.

One exception is that LOGIC_OPT OFF does not prevent a simple register from being optimized into an input pad register. In this case the register node is no longer produced on a macrocell output in the EPLD. To prevent optimization into an input pad register, use the global statement OPTIONS OFF REG_OPT or insert a new node with LOGIC_OPT OFF between the device input and the register.

LOGIC_OPT OFF does not prevent a node from receiving product-terms collapsed in from preceding nodes. In the previous example, using LOGIC_OPT OFF A will not prevent equation B from being collapsed forward into equation A. LOGIC_OPT OFF B will prevent equation B from being collapsed into equation A.

```
LOGIC_OPT ON Q1 Q13 Q24; Optimize only these equations
```

```
LOGIC_OPT OFF; Turn off optimization globally
```

Figure 4-13   LOGIC_OPT Syntax Examples

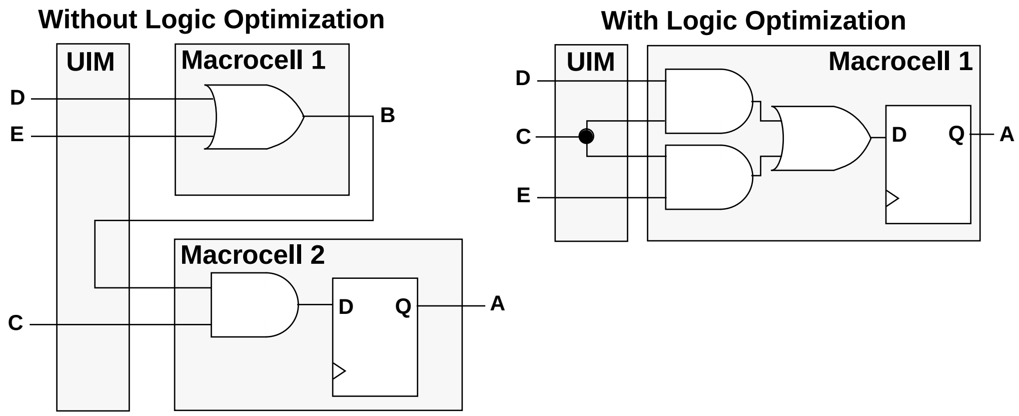


Figure 4-14   Using the LOGIC_OPT Statement

MINIMIZE

[MINIMIZE {OFF | ON} [*signal_name*…]]

Use the MINIMIZE statement in Top-Level Files, Include Files, and PLD files.

The MINIMIZE statement selectively enables or disables Boolean equation minimization. By default, MINIMIZE is enabled for all sum-of-products equations. If you omit the *signal_name* list, the MINIMIZE statement applies to all equations. If you include the *signal_name* list, the specified MINIMIZE setting applies to the listed signals and the opposite setting applies to all other signals. Only one MINIMIZE statement is allowed per equation file.

Minimization can be performed only on signals defined using the standard sum-of-products equation form; equations using the ALU format are never minimized. If MINIMIZE is ON the software selectively creates the DeMorgan equivalent of your equations to produce the least number of product terms.

Equation minimization produces a more compact design requiring fewer device resources.

```
MINIMIZE OFF F
```

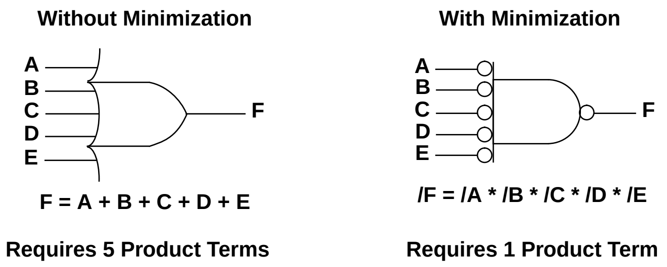


Figure 4-15 Using the MINIMIZE Statement

MRINPUT

`[MRINPUT]`

Use the MRINPUT statement in Top-Level Files only.

The MRINPUT statement allows you to use the Master Reset pin of the XC7354 and XC7336 devices as an additional logic input pin. The MRINPUT statement indicates that the pin is used for a logic input. If there is no MRINPUT statement in your design file, the default uses the pin as the Master Reset input.

```
MRINPUT; Use the Master Reset input for logic
```

Figure 4-16 MRINPUT Syntax Example

NODE

`[NODE [({UIM | NODETRST})] [/] `*`signal_name`*`...]`

Use the NODE statement in Top-Level Files, Include Files, and PLD files.

The NODE statement declares the internal signals that do not appear on device pins. In PLD equation files, custom component equation files, or Include Files, you may only use the NODE statement with the UIM or NODETRST options. Each signal defined as a node must appear on both the left side of an equation and right side of an equation in the design.

If you use the UIM option, each signal name specifies a function to be performed in the UIM. Signals declared as NODE (UIM) should be assigned one-level combinatorial equations only.

Note: Manually assigning equations to the UIM is not recommended for most designs.

The NODETRST option allows you to specify a .TRST equation for a node to disable the macrocell feedback to the UIM. This is used to emulate 3-state bussing within the EPLD device.

```
INPUTPIN A B C E F START
OUTPUTPIN Y0
NODE (UIM) D G
NODE Y0
EQUATIONS
D = /A*B*/C; UIM node
G = /E*F; UIM node
Y0 = D+G+START; macrocell node
```

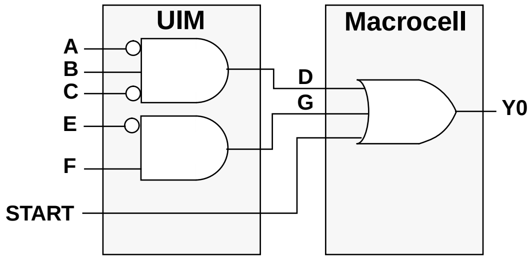


Figure 4-17   Using the NODE (UIM) Statement

OPTIONS

`[OPTIONS OFF [REG_OPT] [CLOCK_OPT] [UIM_OPT] [FOE_OPT]]`

Use the OPTIONS statement in Top-Level Files only.

Use the OPTIONS statement to inhibit various XEPLD optimization routines for your design.

- REG_OPT – Normally allows the software to use the input pad registers wherever possible to reduce the macrocell resource requirements. (Default ON)
- CLOCK_OPT – Normally allows the software to allocate the dedicated FastCLK nets to the simple clock inputs specified in your design. (Default ON)
- UIM_OPT – Normally allows the software to use the UIM AND capability wherever possible to reduce the macrocell resource requirements. (Default ON)
- FOE_OPT – Normally allows the software to allocate the dedicated Fast Output Enable nets to the simple 3-state control inputs in your design. (Default ON)

If you omit the OPTIONS statement, all options default to ON and the software will automatically assign the optional device resources wherever possible to reduce the macrocell resource requirements.

Note: You can also assign the optional resources manually by using the appropriate PLUSASM commands.

```
OPTIONS OFF REG_OPT FOE_OPT; CLOCK_OPT and UIM_OPT
                           ; remain ON
```

Figure 4-18   OPTIONS Syntax Example

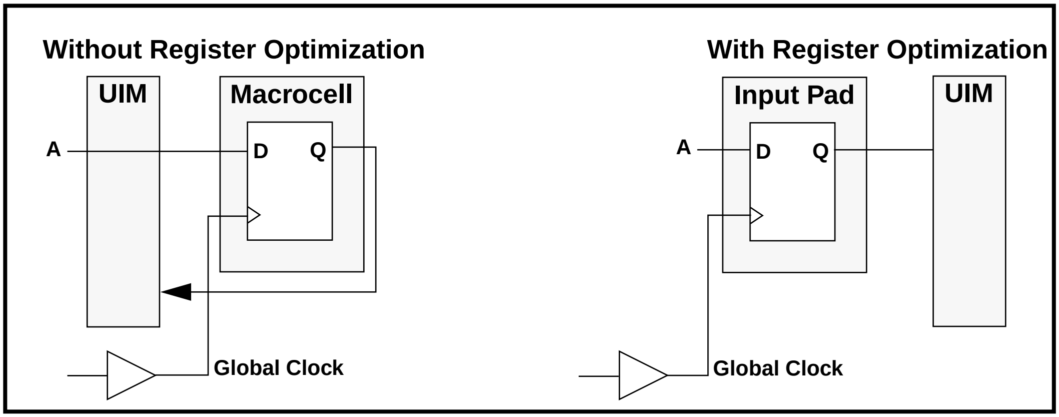


Figure 4-19 Using the REG_OPT Option

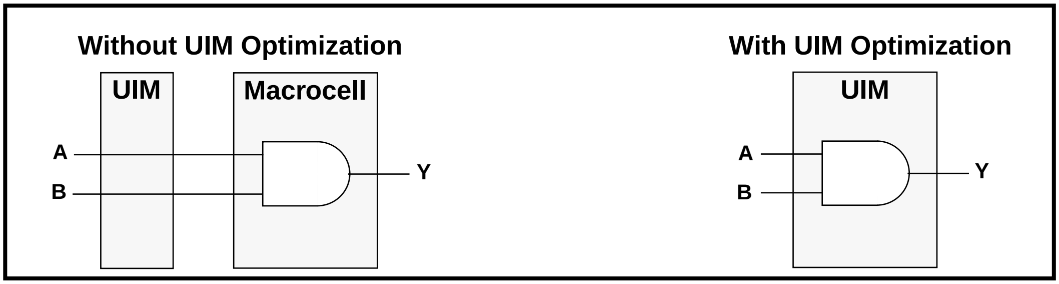


Figure 4-20 Using the UIM_OPT Option

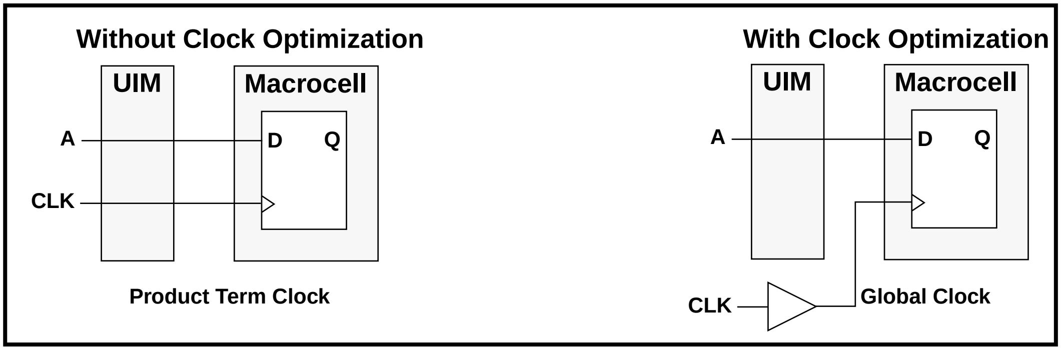


Figure 4-21 Using the CLOCK_OPT Option

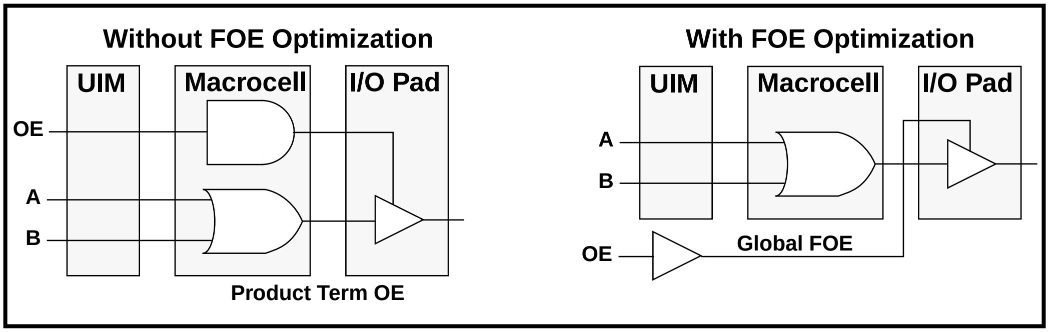


Figure 4-22 Using the FOE_OPT Option

OUTPUTPIN

[OUTPUTPIN [([FOE = ***foe_name*****] [NODETRST])] [/]**
signal_name**... [PIN** ***pin_number*** **...]]**

Use the OUTPUTPIN statement in Top-Level Files only.

The OUTPUTPIN statement declares the signals that drive XC7000 device pins. A design may contain several OUTPUTPIN statements, specifying various combinations of pin options including the following:

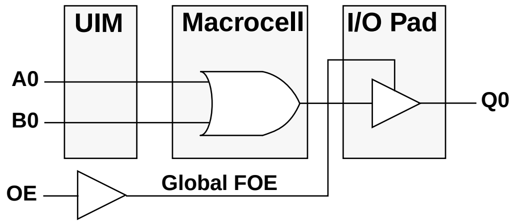
FOE = *foe_name*

Each 3-state output is enabled by *foe_name*. You can use this option in conjunction with *signal_name*.TRST equations for the same outputs. (FOE does not apply to XC7272 designs.)

NODETRST

Any *signal_name*.TRST equations specified for these outputs affect the macrocell feedbacks as well as the device pin drivers. (This is the assumed behavior of XC7272 designs, in which 3-state control always affects both pin drivers and the macrocell feedbacks.)

```
OUTPUTPIN (FOE=OE) Q0
FOEPIN OE
EQUATIONS
Q0 = A0 + B0
```



```
OUTPUTPIN (NODETRST) Q0
EQUATIONS
Q0 = A + B
Q0.TRST = OE
```

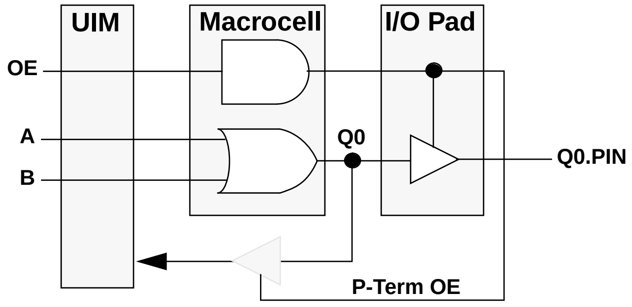


Figure 4-23 Using the OUTPUTPIN Statement

PARTITION

[PARTITION {*partition_name* FB*n* | FB*n_m* | FFB |FB} *signal_name*…]

Use the PARTITION statement in Top-Level Files, Include Files, and PLD files.

The PARTITION statement explicitly controls the placement of equations into physical Function Blocks and macrocells. There are two types: Physical and Logical PARTITION statements.

The PARTITION statement is required only for creating adders (which require contiguous placement of equations) and for assigning speed-critical equations to FFBs. Though you can use the PARTITION statement for pin assignment, it is not recommended; use the PIN specification option in the PLUSASM I/O pin declaration statements instead.

Note: In Include Files use only Logical PARTITION statements. In PAL equation files, use PARTITION statements only for establishing the relative macrocell order of arithmetic and local shift functions.

Note: Do not use PARTITION statements to establish the order of .EXPORT functions in equations targeted to the PLFFB9 component; use the signal list in the CHIP statement to establish the signal order.

Note: Do not use the PARTITION statements to establish the order of local shift and arithmetic functions in the PLFB9 component; use the signal list in the CHIP statement to establish the signal order.

Physical PARTITION Statements

Physical PARTITION statements assign equations to specific Function Blocks or specific macrocells.

Specify FB*n* as the *partition_name* to explicitly map the specified output signal names into the designated Function Block (*n*). For example, to place signal X2 into any macrocell of Function Block 3:

```
PARTITION FB3 X2
```

Specify FB*n_m* as the *partition_name* to explicitly map the specified output signal names consecutively into the designated Function Block (*n*), beginning with macrocell (*m*). For example, to place signals X1, Y1, and Z1 into Function Block 3, macrocells 5, 6, and 7 (respectively) use the following:

```
PARTITION FB3_5 X1 Y1 Z1
```

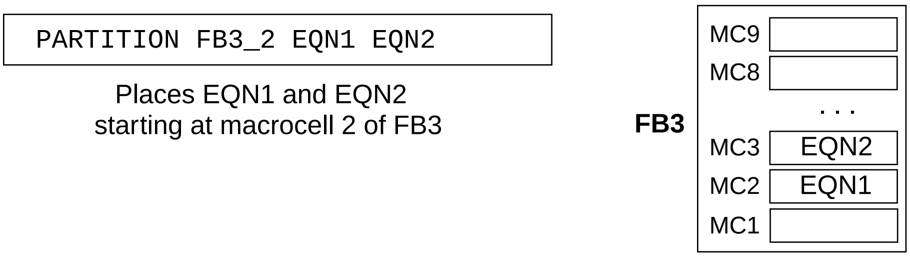

PARTITION FB3_2 EQN1 EQN2
Places EQN1 and EQN2 starting at macrocell 2 of FB3
FB3
MC9
MC8
. . .
MC3
EQN2
MC2
EQN1
MC1

Figure 4-24   Using Physical PARTITION Statements

You can specify any number of signal names in the PARTITION FBn_m statement. Equations will be mapped into consecutive macrocells of consecutive function blocks if necessary.

Logical PARTITION Statements

Logical PARTITION statements are used to keep equations grouped together in the same function block (to share resources) or to designate the type of function block to be used without determining a specific physical location; the software is allowed to choose the most efficient specific physical placement. Logical partitions are also used to specify the relative order in which equations are placed into macrocells when they are linked by arithmetic carry, local macrocell shift, or FFB product term assignment.

Use the PARTITION FB statement to assign outputs to High Density Function Blocks (the outputs are not necessarily placed in the same FB). Use the PARTITION FFB statement to assign outputs to Fast Function Blocks (the outputs are not necessarily placed in the same FFB). For Example:

```
PARTITION FB X Y Z; assign X, Y, Z to any FBs
PARTITION FFB A B C; assign A, B, C to any FFBs
```

Use the PARTITION *partition_name* statement to group a list of outputs into a common Function Block which can be any FB or FFB at the discretion of the software. When grouping equations you may specify no more than 9 outputs in the Logical PARTITION statement. For example, the following statement specifys that F1 through F8 are to remain together in the same function block:

```
PARTITION GROUP1 F1 F2 F3 F4 F5 F6 F7 F8
```

To specifically assign a group of outputs to either an FB or FFB, and keep them together in the same Function Block, you must use two PARTITION statements. One PARTITION statement is used to group the outputs and another is used to assign the group to a specific type of Function Block (either FB or FFB). For example, to assign outputs Q0, Q1, and Q2 to the same Fast Function Block, do the following:

```
PARTITION QGROUP Q0 Q1 Q2; group the outputs
PARTITION FFB Q0 Q1 Q2; assign the group to a FFB
```

In this example, the software will choose a single Fast Function Block in which to place equations Q0, Q1, and Q2.

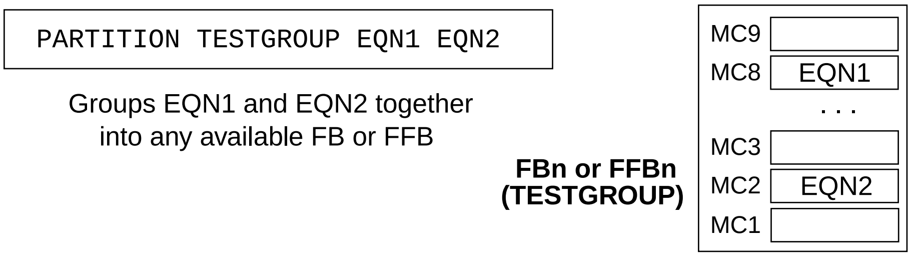

PARTITION TESTGROUP EQN1 EQN2
Groups EQN1 and EQN2 together into any available FB or FFB
FBn or FFBn (TESTGROUP)
MC9
MC8
EQN1
. . .
MC3
MC2
EQN2
MC1

Figure 4-25 Using Logical PARTITION Statements

Linked Equations

When any of the equations specified in a Logical or Physical PARTITION statement participate in an arithmetic carry chain, a local macrocell shift chain (*signal_name*.SHIFT), or an explicit FFB product term assignment chain (*signal_name*.EXPORT), the named equations form a linked list. The equations in a linked list are always mapped in the order in which they are listed into consecutive macrocell locations. You may specify any number of signal names in a linked equation list in any PARTITION statement if the chain can be implemented in the target device architecture. If you specify more than nine signal names in a linked list, the equations are mapped into consecutive macrocells of consecutive function blocks.

For example, to define the carry order of a 12-bit adder, you can allow the software to determine the best starting macrocell location by using the following Logical Partition statement:

```
PARTITION ADD12 Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10
  Q11
```

You could also designate a specific starting macrocell location by using the following Physical PARTITION statement:

```
PARTITION FB7_9 Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10
  Q11
```

Note: The XC7000 architecture does not support FFB p-term assignment across function block boundaries; p-term assignment linked lists are limited to nine equations.

General Rules for PARTITION Statements

- If two or more equations are linked in a PARTITION statement then all equations are kept contiguous in order. For example, if signals Q0, Q1, and Q2 form a 3 bit adder, in the following statement, signals X, Y, and Z will physically remain in the same relative positions listed:

  ```
  PARTITION TEST X Y Q0 Q1 Q2 Z
  ```

- You can only use one PARTITION FBn statement per function block.
- If you use a PARTITION FBn statement you cannot use the PARTITION FBn_m statement for the same function block. However, you may use any number of PARTITION FBn_m statements if the specified macrocells do not overlap.
- Physical partitions using ten or more non-linked macrocells must indicate a specific starting macrocell. For example:

  ```
  PARTITION FB3_5 A1 A2 ... A10
  ```

  The following physical partition syntax will cause an error because a specific macrocell is not indicated:

  ```
  PARTITION FB3 A1 A2 ... A10
  ```

- Logical partitions with ten or more equations must contain linked equations (.add, .shift, .export, .addmode). The following will cause an error if none of the equations are linked:

  ```
  PARTITION TEST A1 A2 ... A10
  ```

- Any conflicts between pin assignments and PARTITION statements cause errors.
- The range of macrocells specified in physical PARTITION statements must not overlap.

Logical Partitions Using Less Than 9 Macrocells

Example: **`PARTITION TEST A B C D E`**

Note: This method is used primarily for arithmetic functions that require only a single function block.

- All macrocells are placed into one FB; macrocell assignments within the function block are determined by the software.
- Non-linked macrocells may be placed anywhere within the FB and not necessarily in contiguous order, at the discretion of the software.
- If the macrocells are linked (using either .add, .shift, .export, or .addmode) the equations are kept contiguous in the listed order; the starting macrocell is determined by the software.

Logical Partitions Using More Than 9 Macrocells

Example: PARTITION TEST A B C D E F G H I J

Note: This method is used primarily for arithmetic functions that require multiple function blocks.

- The macrocells must be linked (using .add, .addmode, or .shift only).
- The equations cannot use FFB product term assignment (.export).
- The equations will always be placed in High Density FBs; not in FFBs
- The first macrocell may be placed anywhere, at the discretion of the software.
- This method can be used for relative pin assignment by including at least one linked equation.

Physical Partitions Using Less Than 9 Macrocells

Example: **`PARTITION FB4 A B C D E`**

- All macrocells are placed into the specified FB (FBn). None of the signals are allowed to flow into an adjacent FB.
- Non-linked macrocells may be placed anywhere within the FB and not necessarily in contiguous order, at the discretion of the software.

- If the macrocells are linked, the equations are kept contiguous in the listed order; the starting macrocell is determined by the software.
- This method can be used for approximate pin assignment.

Physical Partitions Specifying Starting Macrocells

Example: **`PARTITION FB4_5 A B C D E F G`**

- The signal chain always starts on the specified macrocell and the signals are kept contiguous in the listed order for *both* linked (.add, .shift, .export, .addmode) and non-linked equation lists.
- If there are more than 9 macrocells, the next macrocell in the chain is always placed in the same type of function block as the preceding one; high density and Fast Function Blocks are not mixed in the chain.
- This method can be used for exact pin assignment, but the normal pin assignment syntax of the I/O pin declaration statements is recommended.

```
PARTITION FB sig_A; place sig_A in any High Density FB
PARTITION FFB sig_B ; place sig_B in any Fast FB
PARTITION FB2 sig_C sig_D; place sig_C and sig_D in FB2
PARTITION FB3_5 sig_E; place sig_D in FB3, macrocell 5
PARTITION MUX5 X Y; place X and Y in the same partition
          ;(labeled MUX5) in either an FB or FFB
...
PARTITION COUNTER Q0 Q1 Q2 Q3; group the counter
PARTITION FB Q0 Q1 Q2 Q3; place the counter in any HDFB
```

Figure 4-26 PARTITION Syntax Examples

PWR

[PWR {LOW | STD} [*signal_name…*]]

Use the PWR statement in Top-Level Files only.

The PWR statement specifies whether macrocells implementing the specified equations have LOW-power or standard-power (STD) operation. If you omit the *signal_name* list, the PWR statement applies to all macrocells. If you include the *signal_name* list, the listed signals are given the specified setting and all other signals are given the opposite setting.

```
PWR LOW X1 X2 X3; low power used for the these macrocells
```

Figure 4-27 PWR Syntax Example

STRING

[STRING *string_name* '*string_text*']

Use the STRING statement in Top-Level Files, Include Files, and PLD files.

The STRING statement allows you to name any text string and then use the name instead of the string in equations. This can save you time and make your equations easier to read.

```
STRING LONG_EQ ‘A+B+C+D+E+F+G*Q+/X+/Y+/Z’
...
EQUATIONS
IN1 = LONG_EQ + D0
IN2 = LONG_EQ + D1
...
IN8 = LONG_EQ + D7
```

Figure 4-28 STRING Syntax Example

The Equation Section

The equation section of a PLUSASM file is where you define the functionality of your design. You must begin the equation section with the EQUATIONS keyword, which must appear after all declarative statements and before any logic equations.

PLUSASM recognizes three types of equations: Combinatorial, Registered, and Control. The following equation syntax can be used in Top-Level Files, Include Files, and PLD equation files unless otherwise specified.

Combinatorial and Registered Equations

Combinatorial equations create immediate outputs from operations on input signals and are expressed in the following form:

[/] *signal_name* = *expression*

where "/" indicates a signal inversion.

Registered equations specify the logic function that drives the D input of a rising-edge-triggered flip-flop. The format of a registered equation is identical to combinatorial format except that the "colon equal" (:=) assignment operator is used, as in:

[/]*signal_name* := *expression*

The logic-defining expression for a combinatorial or registered equation can take one of two forms.

- The standard sum-of-products format. This form is compatible with PALASM.

  [/]*signal_name* [:]= *p_term* [+ *p_term*]...

  where *p_term* is a logic expression in the form of:

  [/]*input_name* [* [/] *input_name*]...
- The ALU format. This form is specific to PLUSASM and to the XEPLD High-density Function Block architecture. To define output logic using the ALU format, you specify two sum-of-product expressions, named D1 and D2, and the ALU operation which combines them:

```
signal_name.D1={p_term [+ p_term]...| VCC | GND}
signal_name.D2={p_term [+ p_term]...| VCC | GND}
[/]signal_name [:]= signal_name.D1 alu_op
signal_name.D2
```

where alu_op is one of the 16 logical operation keywords as shown below:

Table 4-1 ALU Function Keywords

Keyword	Logical Operation	Keyword	Logical Operation
XOR	F = D1 :+: D2	**D1_ONLY**	F = D1
XNOR	/F = D1 :+: D2	**D2_ONLY**	F = D2
AND	F = D1 * D2	**NOTD1**	F = /D1
NAND	/F = D1 * D2	**NOTD2**	F = /D2
OR	F = D1 + D2	**D1_AND_NOTD2**	F = D1 * /D2
NOR	/F = D1 + D2	**NOTD1_AND_D2**	F = /D1 * D2
VCC	F = 1	**D1_OR_NOTD2**	F = D1 + /D2
GND	F = 0	**NOTD1_OR_D2**	F = /D1 + D2

Note: In PLUSASM Syntax:

- The :+: symbol is the same as XOR.
- The * symbol is the same as AND.
- The + symbol is the same as OR.

The three-line ALU format may be used as an alternative to the standard PALASM-compatible equation format for any High Density Function Block output function.

The polarity of an output function expressed in ALU format is determined by the ALU operation. None of the three equations comprising the ALU format may be preceded by a slash (“/”).

Refer to the appropriate EPLD device data sheet for the allocation of D1 and D2 product terms among macrocells and Function Blocks.

Note: As an alternative to the three-line ALU format, XOR equations can also be specified by using the in-line :+: operator. For example:

```
D.ADD = VCC
D := A :+: B
```

```
INPUTPIN A0 A1 SEL CLEAR
OUTPUTPIN S1
FASTCLOCK CLK
EQUATIONS
S1.D1 = A0 * /SEL + A1 * SEL
S1.D2 = S1 * /CLEAR
S1 := S1.D1 XOR S1.D2
```

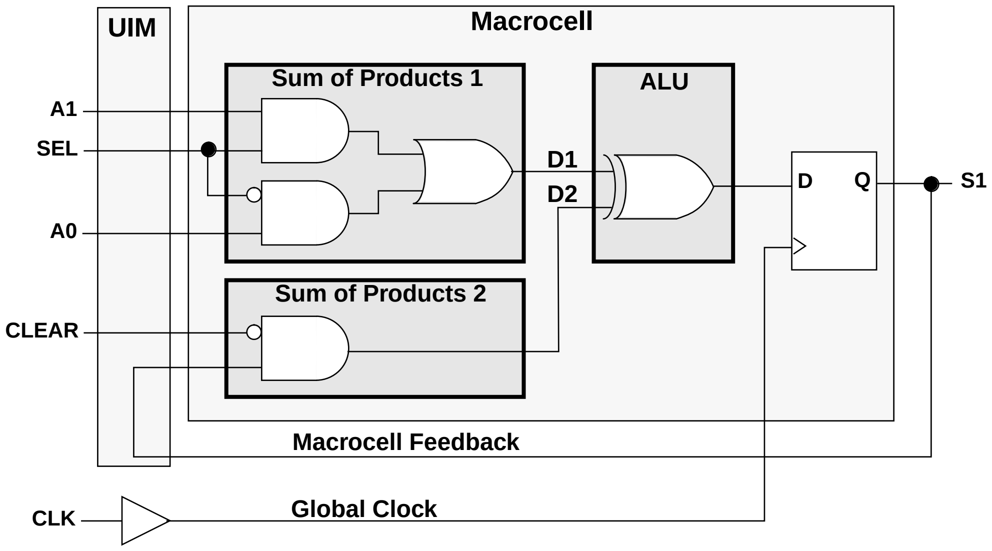


Figure 4-29 Equation Syntax Example

Control Equations

You can use control equations to describe additional control functions for combinatorial or registered outputs. Control equations begin with the output name used in the associated logic-defining equation, and are appended with an extension keyword. For example:

```
signal_name.extension = expression
```

The polarity of the control equation itself cannot be inverted (except for the .T extension); any slash (/) placed before the equation is ignored. Control equations override any corresponding default.

The control equation extensions are:

- **.ADD** (Enables arithmetic carry mode — not used with XC7272 or XC7336.)
- **.ADDMODE** (Enables arithmetic carry mode — XC7272 only.)
- **.CLKF** (Specifies clock control source for registered outputs.)
- **.D1** and **.D2** (Specifies the signal inputs to the ALU — not used with the XC7336.)
- **.EXPORT** (Passes p-terms from one macrocell to another — not used with the XC7272 or XC7236.)
- **.FBK** (Enables local feedback, ORed with the D2 ALU input — not used with the XC7272 or XC7336.)
- **.FBKINVERT** (Inverts the macrocell feedback — used only by the software, not required by the user.)
- **.FI** (Specifies the signals taken from the FastInput pins — not used with the XC7272 or XC7236.)
- **.PIN** (Specifies the signals taken from device pins instead of macrocell feedback.)
- **.PRLD** (Specifies the initial preload value of registers — not used with the XC7336.)
- **.RSTF** (The asynchronous reset control function.)
- **.SETF** (The asynchronous set control function.)
- **.SHIFT** (Enables feedback from the previous HDFB macrocell, ORed with D2 ALU input — not used with XC7272 or XC7336.)
- **.T** (The toggle flip-flop function. A preceding "/" is allowed)
- **.TRST** (The 3-state output enable function.)

.ADD (Arithmetic Carry Enable)

```
[signal_name.ADD = VCC]
```

The .ADD extension enables arithmetic mode (arithmetic carry-in) for the ALU block of all XC7000 series devices except the XC7272 and trhe XC7336. When you enable the add mode, the carry produced by the previous macrocell is XORed with the result from the ALU operation on the D1 and D2 intermediate sum-of-products, as defined by the *alu_op*. You may either use the 3-line ALU equation format or the more convenient in-line XOR operator (:+:) when using arithmetic mode. For example:

```
D.ADD = VCC
D := A :+: B
```

Note: The .ADD equation enables the carry-in signal into the current macrocell. For PLFB9 equation files, PLUSASM determines the order in which to map the outputs from their order in the signal list. For equation files other then the PLFB9 and for behavioral designs, outputs participating in an arithmetic carry chain must be grouped together with a PARTITION statement; the outputs are mapped in the order in which they appear in the PARTITION statement. The carry-in signal for each output originates from the output whose name immediately precedes it in the signal list.

```
INPUTPIN A1 B1
OUTPUTPIN S1
EQUATIONS
S1 = A1 :+: B1
S1.ADD = VCC
```

Figure 4-30   .ADD Syntax Example

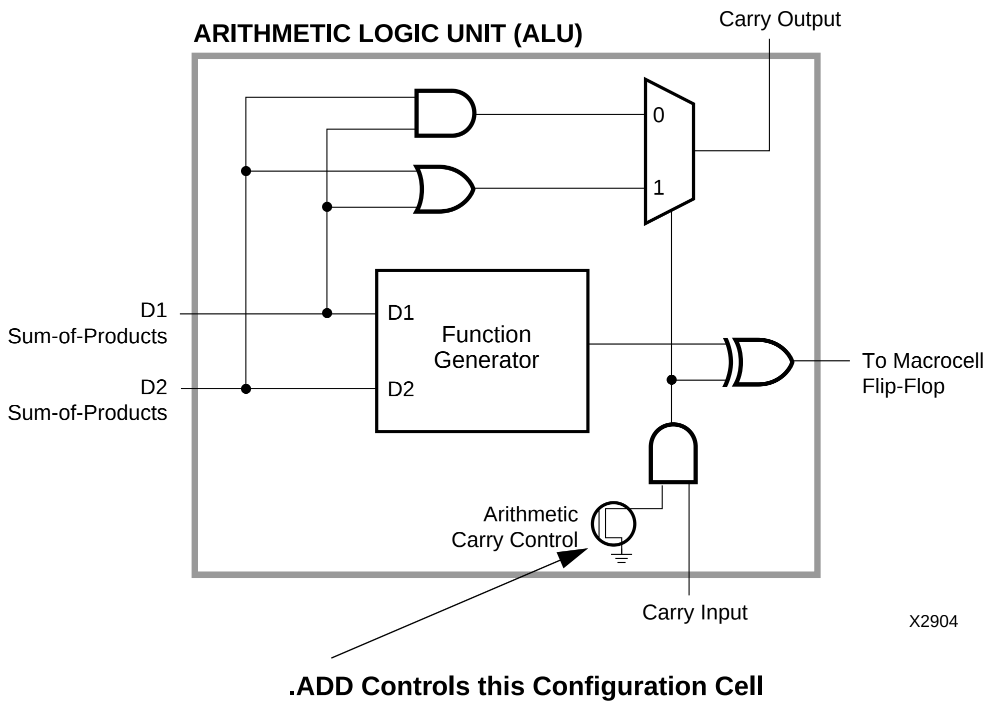


Figure 4-31   Using the .ADD Extension

.ADDMODE (Arithmetic Carry Enable)

[*signal_name*.ADDMODE = VCC]

The .ADDMODE extension enables the arithmetic mode (arithmetic carry-in and carry-out) for the ALU block of the XC7272 only. When you enable the add mode, the macrocell output value is derived from the D1 sum-of-products and the carry-in from the previous macrocell, according to the ALU operation specified. The D2 sum-of-products is used only to generate the carry-out. You must use the 3-line ALU equation format when using arithmetic mode as shown in Figure 4-32.

The .ADDMODE equation enables the carry-in and carry-out signals of the current output. For PLD equation files, PLUSASM determines the order in which to map the outputs from their order in the signal list. For behavioral designs, outputs participating in arithmetic carry chains must be grouped together with PARTITION statements; the outputs are mapped in the order in which they appear in the PARTITION statement. The carry-in signal for each output originates from the output whose name immediately precedes it in the signal list.

Note: The D1 and D2 functions of the XC7272 are not interchangeable.

```
INPUTPIN A1 B1
OUTPUTPIN S1
EQUATIONS
S1.D1 = A1 * /B1 + /A1 * B1; Half Adder = A1 XOR B1
S1.D2 = B1; Carry out = B1 when S1.D1 is false
S1 := S1.D1 XOR S1.D2; using 1 of 16 local operands
S1.ADDMODE = VCC; Enable the fast carry chain
```

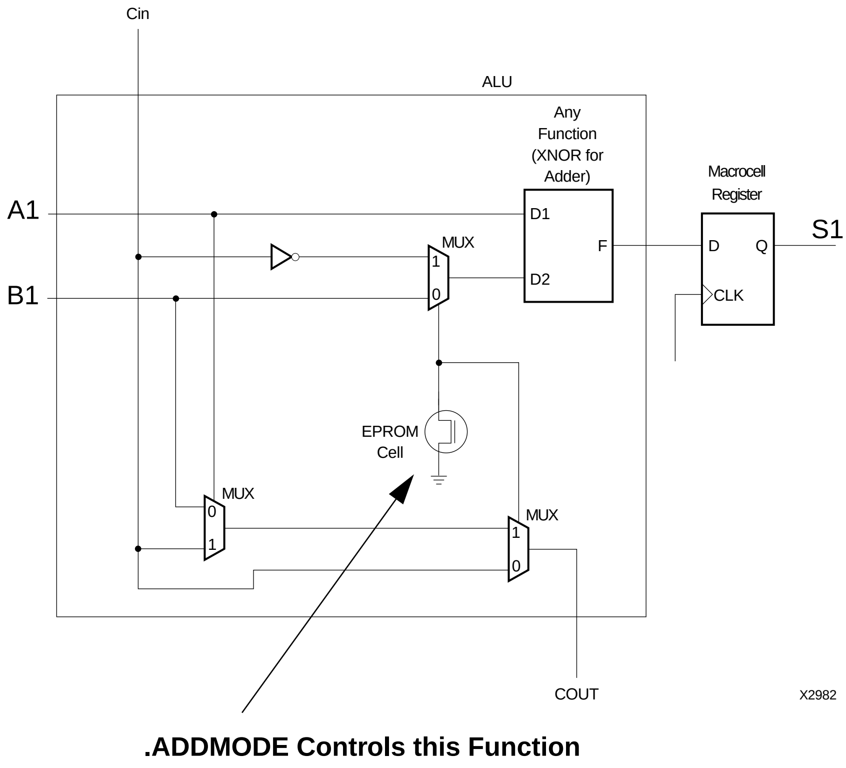


Figure 4-32 Using the .ADDMODE Extension

.CLKF (Register Clock Source)

```
[signal_name.CLKF= {p_term | fastclock_name
|GND}]
```

The .CLKF extension specifies the clock control source for a registered output. If a FastCLK signal is used, its signal name must appear alone and uninverted in the .CLKF equation. If a product term is specified (for a High-density Function Block only), the register clock input goes active whenever the value of the product term expression changes from false to true. GND specifies that the register changes only in response to .SETF and .RSTF functions (for a High-density Function Block only).

The .CLKF equation for a registered output may be omitted only if a default clock signal has been specified in a FASTCLOCK statement in a Top-Level File.

Note: If there is no .CLKF statement, the first FASTCLOCK is the default clock source for the equation.

Note: Default clocks for PLDs can be either FastCLKs or a product-term clocks.

```
INPUTPIN D0 D1 CLK0
OUTPUTPIN Y0 Y1
FASTCLOCK CLK1
EQUATIONS
Y0 := D0
Y0.CLKF = CLK0; Input and I/O uses P-Term clock

Y1 := D1; CLK1 is implied (because no .CLKF statement)
```

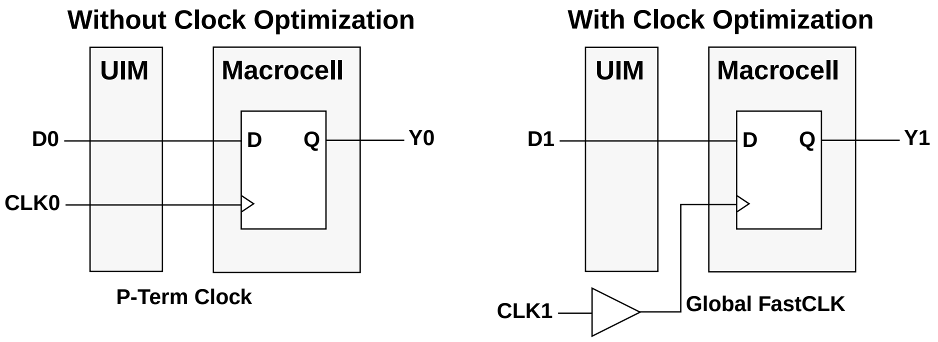


Figure 4-33 Using the .CLKF Extension

.D1 and .D2 (ALU Inputs)

```
[[signal_name.Dl = {p_term [+ p_term]... | VCC |
GND}]
[signal_name.D2 = {p_term [+ p_term]... | VCC |
GND}]
signal_name [:]= signal_name.D1 alu_op
signal_name.D2]
```

The .D1 and .D2 extensions assign signals to the D1 and D2 inputs of the ALU. See the device data sheets for details of the D1 and D2 input architecture.

There are 16 possible ALU operations that can be performed on the D1 and D2 inputs as shown below:

Table 4-2 ALU Function Keywords

Keyword	Logical Operation	Keyword	Logical Operation
XOR	F = D1 :+: D2	D1_ONLY	F = D1
XNOR	/F = D1 :+: D2	D2_ONLY	F = D2
AND	F = D1 * D2	NOTD1	F = /D1
NAND	/F = D1 * D2	NOTD2	F = /D2
OR	F = D1 + D2	D1_AND_NOTD2	F = D1 * /D2
NOR	/F = D1 + D2	NOTD1_AND_D2	F = /D1 * D2
VCC	F = 1	D1_OR_NOTD2	F = D1 + /D2
GND	F = 0	NOTD1_OR_D2	F = /D1 + D2

Using the ALU

The ALU can perform both logical and arithmetic operations. A block diagram of a typical XC7000 family ALU is shown in Figure 4-31.

In logic mode, the ALU performs as a 2-input function generator that implements any of the functions listed in Table 4-2.

In Arithmetic mode the ALU can generate the sum or difference of the D1 and D2 inputs. Combined with the carry input from the previous macrocell, the ALU operates as a 1-bit full adder, generating a carry output to the next macrocell in the chain. The carry chain propagates between adjacent macrocells and also crosses the boundaries between function blocks. The .ADD equation extension enables the carry input to the ALU.

Note: The XEPLD software automatically uses the logic capability of the ALU to implement your design. However, you can also manually specify ALU operations by using the .D1 and .D2 equations.

```
INPUTPIN A0 A1 SEL CLEAR
OUTPUTPIN s1
FASTCLOCK CLK
EQUATIONS
S1.D1 = A0 * /SEL + A1 * SEL
S1.D2 = S1 * /CLEAR
S1 := S1.D1 XOR S1.D2
```

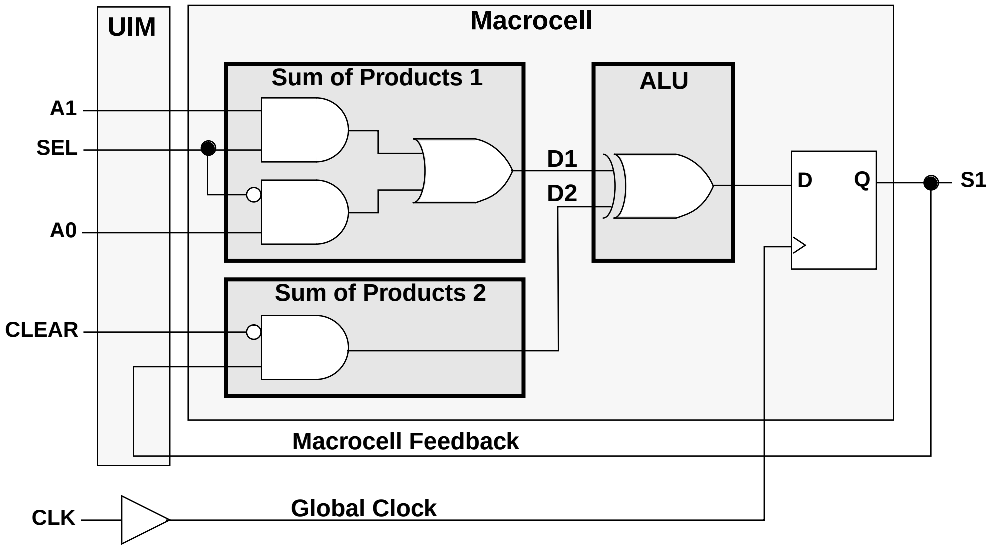


Figure 4-34 Using the .D1 and .D2 Extensions

.EXPORT (FFB Product Term Assignment)

```
[signal_name.EXPORT = p_term [+ p_term]...]
```

The .EXPORT extension specifies a sum-of-products expression of up to four product terms, to be exported and ORed into the next output function (macrocell) of an XC7300-series Fast Function Block. The receiving macrocell may in turn export its four product terms combined with all product terms it received. The donor macrocell (*signal_name*) may still be used as a single product term output (registered or combinatorial) using the equation:

/signal_name* [:]= *p_term

In PLFFB9 equation files, the order of the output signal names in the equation file pin list defines the macrocell order. In behavioral designs, the order of the output signal names in the PARTITION statement (used to define FFB outputs) defines the macrocell order.

Note: FFB equations must use active-low outputs. If you specify an active-high output equation, the software will create the necessary input and output inversions to create an active-low output and maintain your logical functionality.

```
FASTINPUT A B C E F G; Fast input assignment
OUTPUTPIN Y1
NODE Y0
PARTITION FB1 Y0 Y1; Assigns Y0 and Y1 to the same FFB
EQUATIONS
Y0.EXPORT = A + B + C; enables Y0 P-Term assignment
Y1 = E+F+G
```

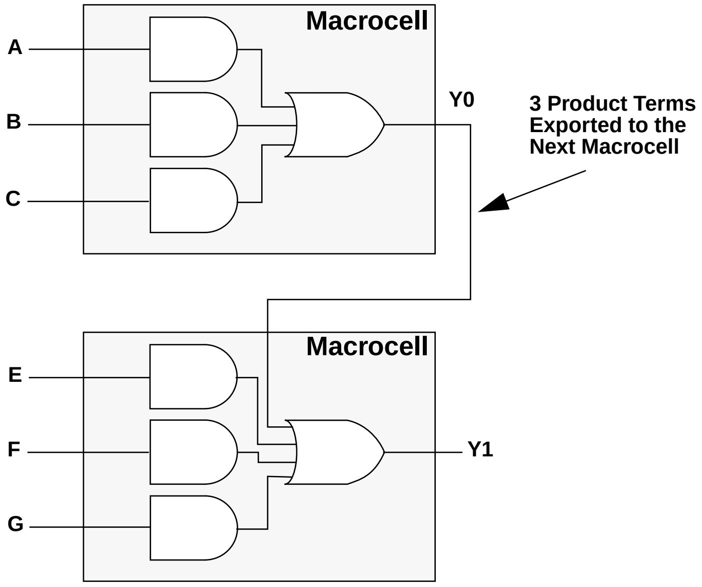


Figure 4-35   Using the .EXPORT Extension

.FBK (Local Feedback)

[*signal_name*.FBK = {*p_term* [+ *p_term*]... | VCC}]

The .FBK extension enables the local feedback of an output to be ORed into the D2 sum-of-products of the same macrocell without pasing through the UIM. When you use .FBK, the local feedback is ANDed with the sum of up to three D2 product terms as specified in the .FBK equation. The output of the AND function is then ORed with the remaining D2 product terms to form the D2 input signal to the ALU. Refer to the EPLD device data sheet for a detailed description of the local feedback logic in the macrocell.

VCC specifies that the local feedback signal is always enabled and ORed into the D2 sum-of-products.

Note: The .FBK extension does not apply to XC7272 or XC7336 designs.

Note: The .FBK extension can be used to increase the available device resources by reducing the number of UIM inputs.

```
...
EQUATIONS
OUT.D2 = A+B+C+D+E+F
OUT.FBK = G+H+I
```

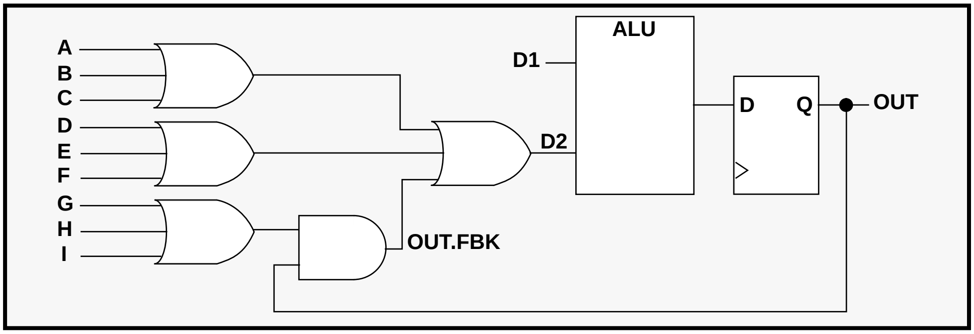


Figure 4-36 Using the .FBK Extension

Logically the D2 product term in this example may be described by the following equation:

D2 = A+B+C+D+E+F+OUT*(G+H+I)

.FBKINVERT (Invert Macrocell Feedback)

[*signal_name*.FBKINVERT = VCC]

The .FBKINVERT extension specifies that the feedback from the macrocell output is to be inverted. This function is performed automatically by the software, as needed, and is never specified by the user.

Note: You should never specify .FBKINVERT. However, the software uses this function when it minimizes your equations and therefore you will see this extension in the *design_name*.EQN file.

```
EQUATIONS
FB1.FBKINVERT = VCC; invert the Feedback signal
```

Figure 4-37 .FBKINVERT Syntax in the .EQN File

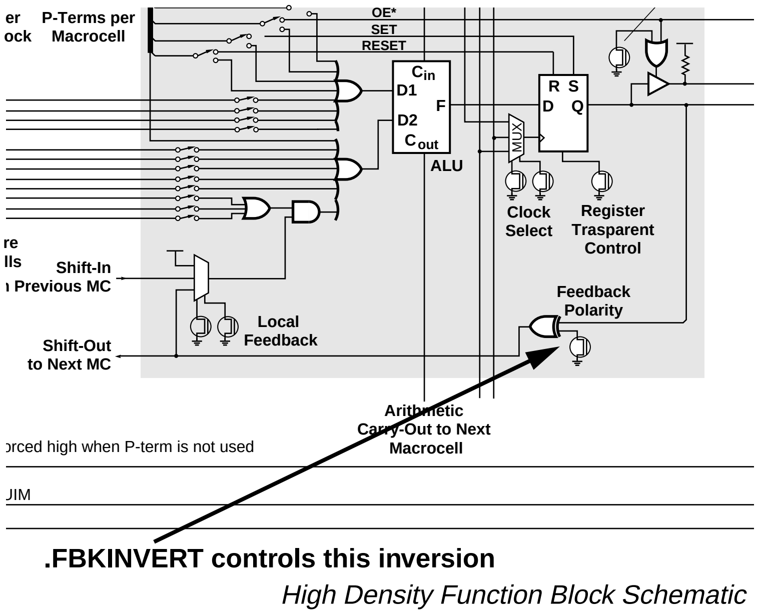

P-Terms per Macrocell
OE*
SET
RESET
Cin
D1
F
D2
Cout
ALU
R S
D Q
MUX
Clock Select
Register Trasparent Control
Shift-In
Previous MC
Local Feedback
Feedback Polarity
Shift-Out to Next MC
Arithmetic Carry-Out to Next Macrocell
orced high when P-term is not used
UIM
.FBKINVERT controls this inversion
High Density Function Block Schematic

Figure 4-38 Using the .FBKINVERT Extension

.FI (FastInput Source)

[*input_name*.FI]

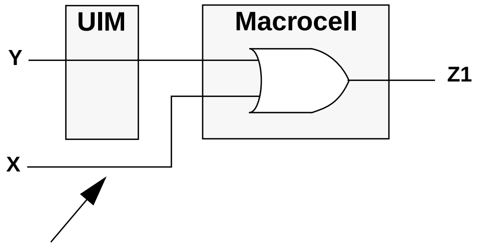
The .FI extension specifies that an input signal is taken from the FastInput path of a FastInput pin (for XC7300-series devices only) rather than from the UIM input path of the same pin. Use this extension only for signal names defined in INPUTPIN or IOPIN statements.

For example, you can use the following equations to select between the signal on pin A (A.FI*DIRECT) and the value latched into the input-pad register which is conditionally loaded from pin A (A*/DIRECT):

```
INPUTPIN A
Q := A.FI*DIRECT + A*/DIRECT
```

Note: The A*/DIRECT signal is available via the UIM path from input A.

```
INPUTPIN X Y
OUTPUTPIN Z1
EQUATIONS
Z1 = X.FI + Y
```



X Bypasses the UIM Using the FastInput Path

Figure 4-39 Using the .FI Extension

.PIN (PIN Input Source)

[*signal_name*.PIN]

The .PIN extension specifies that an input signal is to be taken from a device I/O pin rather than from internal macrocell feedback. Use this expression only for signal names defined in IOPIN statements.

For example, you can use the following equation to conditionally load a macrocell register from a bidirectional pin named Q:

```
Q := Q.PIN*LOAD + Q*/LOAD
```

```
INPUTPIN A B LOAD
IOPIN Y0 Y1
FOEPIN OE
EQUATIONS
Y0 := A + B
Y0.TRST = OE
Y1 = Y0.PIN + LOAD
```

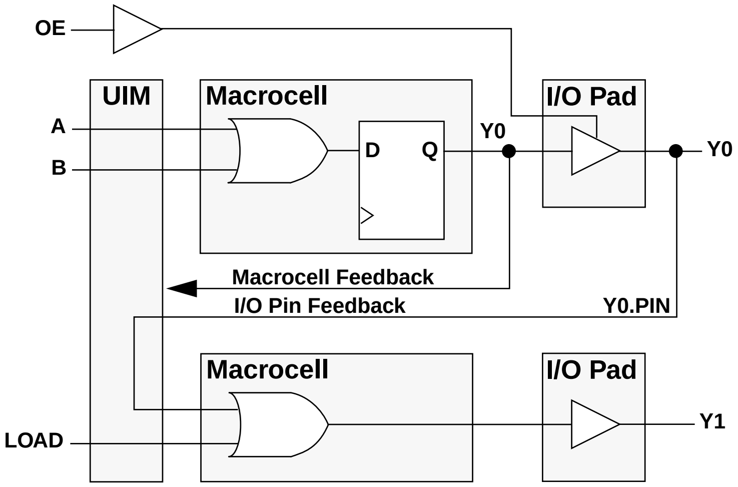


Figure 4-40 Using the .PIN Extension

.PRLD (Preload State)

[*signal_name*.PRLD = {VCC | GND}]

The .PRLD extension specifies the initial logic value to be pre-loaded into an output register during power-up (VCC = 1, GND = 0). The equation output polarity is adjusted before the register and does not effect the output level resulting from a .PRLD operation.

If PRLD is not specified, the software is free to determine the optimal pre-load value for the implementation. The default pre-load value is low for High Density Function Block outputs but may change due to optimization into input-pad registers or FFBs, or by optimization of nearby inverters.

Note: Using the .PRLD extension can prevent the software from using input registers.

Note: The .PRLD extension is not allowed for the XC7336 or any Fast Function Block Equations.

```
EQUATIONS
TEST1.PRLD = VCC; preload the TEST1 output with a logic 1
TEST2.PRLD = GND; preload the TEST2 output with a logic 0
```

Figure 4-41 .PRLD Syntax Example

.RSTF (Asynchronous Reset)

[*signal_name*.RSTF = *p_term*]

The .RSTF extension specifies the asynchronous reset control function for a registered output. The register is forced to and held in the 0 state while the *p_term* product term is true. The equation output polarity is adjusted before the register and does not effect the output level resulting from a .RSTF operation.

```
INPUTPIN A B D0 CLK
OUTPUTPIN Y0
EQUATIONS
Y0 := D0
Y0.CLKF = CLK
Y0.RSTF = A*B; P-Term reset
```

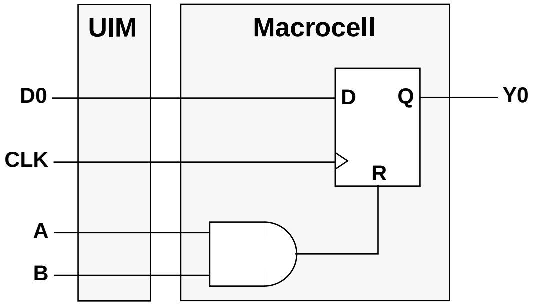


Figure 4-42 Using the .RSTF Extension

.SETF (Asynchronous Set)

[*signal_name*.SETF = *p_term*]

The .SETF extension specifies the asynchronous set control function for a registered output. The register is forced to and held in the 1 state while the *p_term* product term is true. The equation output polarity is adjusted before the register and does not effect the output level resulting from a .SETF operation.

```
INPUTPIN A B D0 CLK
OUTPUTPIN U0
EQUATIONS
U0 := D0
U0.CLKF = CLK
U0.SETF = A*B; P-Term set
```

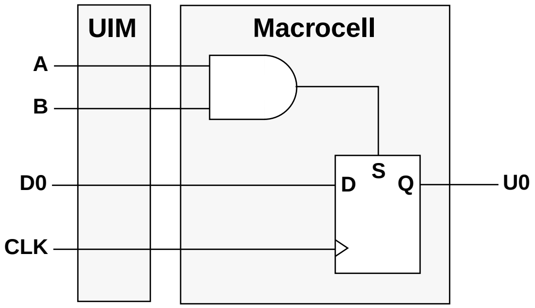


Figure 4-43   Using the .SETF Extension

.SHIFT (Local Shift)

```
[signal_name.SHIFT = { p_term [+ p_term]… |
VCC}]
```

When you use the .SHIFT extension, the output of the previous macrocell is ANDed with the sum of up to three D2 product terms you defined in the SHIFT statement. The output from the AND function is then ORed with the remaining D2 product terms to form the D2 input signal to the ALU. Figure 4-44 shows a diagram of the .SHIFT logic in the macrocell.

VCC specifies that the local feedback signal is always enabled and ORed into the D2 sum-of-products.

Note: The .SHIFT extension does not apply to XC7272 or XC7336 designs.

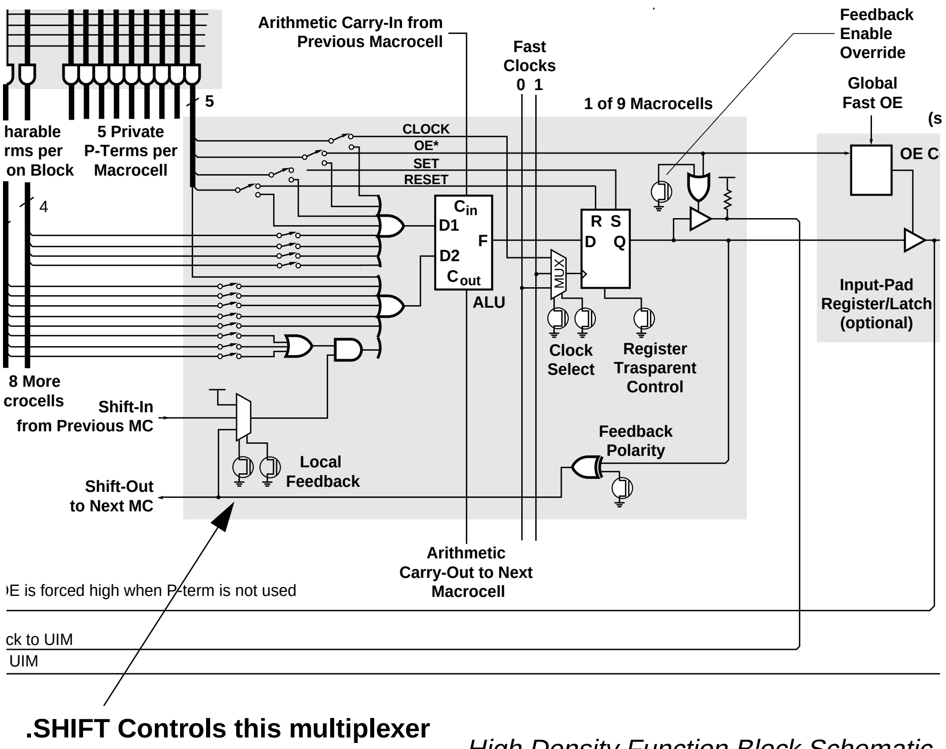


Figure 4-44 The .SHIFT Logic Diagram

```
OUTPUTPIN S1 S0
PARTITION FB S0 S1; Assign the macrocells to the same FB
;                   NOTE: S0 must precede S1
EQUATIONS
S1.SHIFT = P1 + P2 + P3
S1.D1 = P6
S1.D2 = P4 + P5
S1 := S1.D1 XOR S1.D2
...
```

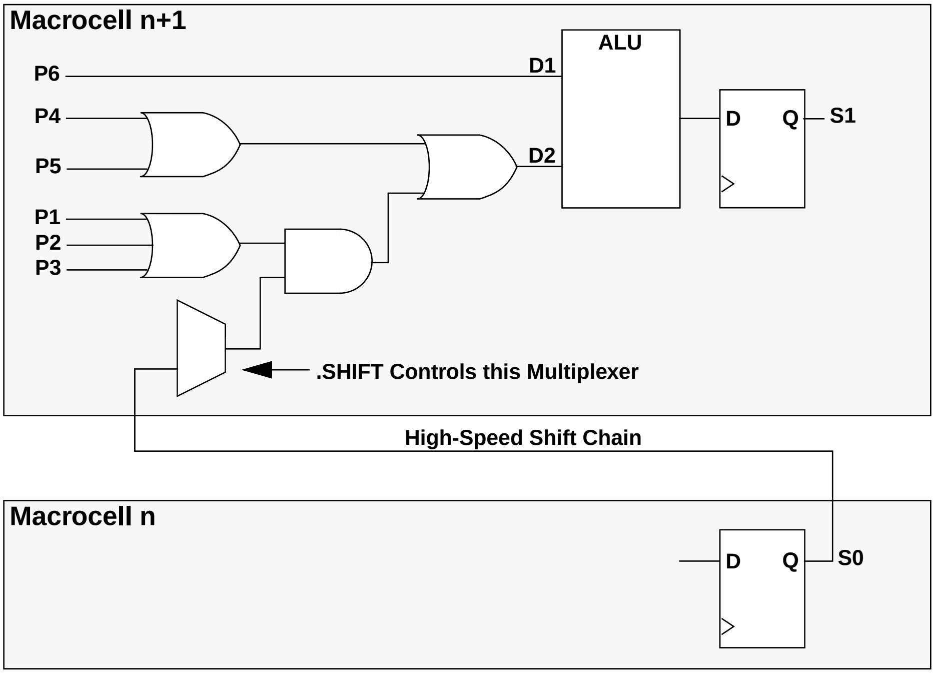


Figure 4-45 Using the .SHIFT Extension

.T (Toggle Flip-Flop Specification)

[[/]*signal_name*.T = *p_term* **[+** *p_term***] ...]**

The .T extension specifies a toggle flip-flop. For the XC7336 the equation is implemented using the Fast Function Block T-flip-flop. For all other devices the equations are implemented as follows:

```
X.T = P1 + P2
```

This is the same as:

```
X := X.D1 XOR X.D2
X.D1 = P1 + P2
X.D2 = X
```

Note: The "=" symbol is used to specify a .T flip-flop instead of the usual ":=" symbol used for other registered equations.

```
INPUTPIN P1 P2
OUTPUTPIN Y0
FASTCLOCK CLK
EQUATIONS
Y0.T = P1 * P2
Y0.CLKF = CLK
```

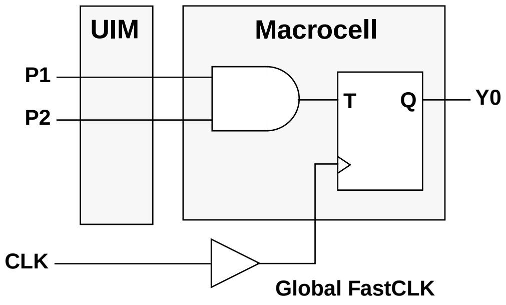


Figure 4-46   Using the .T Extension (XC7336)

.TRST (3-State Control)

[*signal_name*.TRST = {*p_term* | VCC}]

The .TRST extension specifies the output enable condition for a 3-state output signal. The output signal is enabled while the product term *p_term* is true. By default, the 3-state control applies only to the EPLD device pin driver.

Normally, if macrocell feedback is used, and you disable the 3-state output, the macrocell feedback will not be disabled. However, if a node name is declared with NODETRST (or you are targeting the XC7272 which only provides NODETRST behavior), the .TRST equation affects the macrocell feedback as well. By disabling macrocell feedback, you can emulate 3-state bussing within the EPLD device.

```
INPUTPIN A B D0
OUTPUTPIN Y0
EQUATIONS
Y0 := D0
Y0.CLKF = CLK
Y0.TRST = A*B; P-Term 3-state control
```

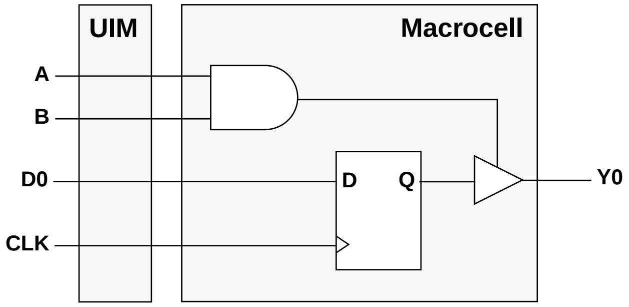


Figure 4-47 Using the .TRST Extension

Defining Signal Polarity in Equations

The interpretation of input and output signal polarities in PLUSASM is identical to standard PALASM syntax.

Output Signal Polarity

You can place a slash,"/", before an output signal name in the signal list of a PLD equation file or in a pin declaration statement of a behavioral design to globally define the signal as active-low. You can also place a slash before an output name in a standard logic equation which defines the condition under which the signal is de-asserted.

PLUSASM implements the output polarity in the device according to both the polarity on the left side of the equation, and the polarity of the corresponding signal name in the signal list (or pin declaration). If these polarities are alike, then the output level will be high when the sum-of-products equation is true (satisfied). If the polarities are opposite, then the output will be low when the sum-of-products equation is true.

```
INPUTPIN A B C D
OUTPUTPIN Y0 Y1
EQUATIONS
Y0 = A + B; Active-HIGH output polarity
/Y1 = C + D; Active-LOW output polarity
Y1.TRST = Y0; Y1 is enabled when Y0 is active
```

Figure 4-48 Using Output Signal Polarity Control

Note: Do not use a slash "/" before the equation name of a control equation such as .TRST, .SETF, .CLKF, .RSTF and so on.

Input Signal Polarity

You can place a slash before an input name in the signal list (or pin declaration statement) to globally define the signal as active-low. You may also place a slash before the input name within the logic expression of an equation to represent the input signal when it is deasserted.

A product term references the high-level version of an input when the equation polarity matches the polarity in the signal list and references the low-level version of the input when the polarities are opposite. This applies to any type of equation and to both High-density FBs and FFBs.

```
INPUTPIN A0 /AS
OUTPUTPIN Y0
EQUATIONS
Y0 = A0 * AS; Polarity inversion occurred in the pinlist
```

Figure 4-49 Using Input Polarity Control

PLUSASM Syntax

This section describes the basic rules for using PLUSASM.

Notation

The syntax notation conventions used in this chapter are as follows.

[ ] **Square brackets** are used to indicate an optional field.

{ } **Braces** indicate a set of mutually exclusive alternatives.

| **Vertical bars** are used to separate alternatives enclosed in braces.

… **Ellipses** indicate one or more occurrences of the preceding field.

italic_type is used to denote a user-specified field.

Square brackets "[ ]", braces "{ }", and vertical bars " | " are not part of the PLUSASM syntax and are never used in a PLUSASM file; they are only used for describing the syntax. However, all parenthesis "( )" appearing in PLUSASM syntax should be interpreted literally and are included in the statement.

Delimiters

Use delimiters to separate adjacent names and key words, or to improve readability.

Spaces — serve as general-purpose delimiters. Consecutive spaces act as one delimiter.

Tabs — are interpreted as spaces.

Return — The end-of-line (return) character terminates a comment field beginning with a single semicolon. You can continue a long statement to the following line or you can write multiple line statements for clarity.

Operators and Special Characters

Equation operators are specified using the special characters listed in Table 5-1. The order of precedence is:

1. The **NOT** operator (/) is processed first.
2. The **AND** operator (*) is processed next.
3. The **OR** operator (+) is processed last.

Table 4-3 Special Characters Used in PLUSASM Syntax

	Character Name	Function
/	Slash	Active low signal or logical NOT
*	Asterisk	Logical AND
+	Plus sign	Logical OR
:+:	Colon plus colon	Logical XOR
=	Equal sign	Combinatorial output assignment
:=	Colon equal	Registered output assignment
,	Comma	Alternate delimiter in signal list
.	Period	Equation extension separator
;	Semicolon	Start of comment (to end-of-line)
;;	Double semicolon	Start & end of multi-line comment

Comments

Start a single line comment at any position on a line with a single semicolon (;). All characters up to the end of the line are comments.

Start a multi-line comment with double semi-colons (;;). All characters up to the next double semicolons are comments.

Reserved Characters

The following characters are reserved and should not be used:

```
` ~ ! @ # $ % ^  { } [ ] " ? |
```

Names

Use only the alphanumeric characters (A-Z, 0-9) and the underscore character (_) in signal names; spaces and tabs are not allowed.

Reserved Words

The following keywords are reserved and may not be used as names in the design:

AUTHOR	INCLUDE_EQN	PARTITION
CEPIN	INPUTPIN	PATTERN
CHIP	INTEGER	PIN_ATTRIBUTE
COMPANY	IOPIN	PINTRST
CONDITIONS	LOGIC_OPT	PWR
DATE	MINIMIZE	REVISION
EQUATIONS	MRINPUT	SIMULATION
FASTCLOCK	NC	STATE
FASTINPUT	NODE	STRING
FOEPIN	NODETRST	TITLE
GND	OPTIONS	VCC
GROUP	OUTPUTPIN	

Unsupported PALASM Syntax

PLUSASM is compatible with most PALASM syntax. The exceptions are described as follows:

Parenthesis

PLUSASM accepts the following usage of parenthesis:

- Parenthesis that enclose the entire right side of an equation:

  ```
  c = (a + b * c :+: d)
  ```

- Parenthesis on either side of an XOR operator:

  ```
  c = (a + b * c) :+: (d)
  ```

- Parenthesis enclosing the entire right side of an equation that contains parenthesis on either side of the XOR operator (a combination of the above two methods):

  ```
  c = ((a + b * c) :+: (d))
  ```

PLUSASM cannot accept the following usage of parenthesis:

- Nested parenthesis:

  ```
  x = ((a + (b *c)))
  ```

- Parenthesis on the left side of an equation:

  ```
  (b = a + c)
  ```

- Parenthesis that do not enclose the entire right side of an equation:

  ```
  z = (a + b) * c + d
  ```

State Machine Syntax

PLUSASM cannot accept state machine syntax such as:

- **%** (don't-care operator).
- **->** (state transition operator).
- **+->** (state transition operator).

The Latched Output Equation Operator

PLUSASM cannot accept the latched output equation operator:

- *=(latched output operator).

Simulation Control Statements

PLUSASM ignores all PALASM simulation control statements (such as TRACE_ON, PRELOAD, CHECK, and so on).

Device Specific Syntax

PALASM includes the capability to write equation files with special (non-standard) syntax for some devices. PLUSASM can accept special syntax only for the 22V10 and 20V8.

For example, the special syntax for the following devices is not recognized by PLUSASM:

- PAL10H20G8.
- PAL32R16.
- PAL32VX10.

If you are converting equation files for these devices you must manually re-write the equations to explicitly define the special or implied functions using standard PLUSASM equation syntax. For a complete list of supported PALs, see Appendix C.

PLUSASM Command Syntax Quick Reference

The following tables provide a quick reference for PLUSASM command syntax. Equation syntax is punctuated as follows:

- Brackets enclose optional fields: [*optional_field*]
- Braces enclose mandatory choices: {*choice1* | *choice2*}
- Vertical bars indicate exclusive options: *choice1* | *choice2*
- Elipses indicate repeatable fields: *repeatable_field* . . .
- Italic type indicates variables: [TITLE *text*]
- Parenthesis are always taken literally, and are included in the equation.

Declaration Statements Used in Schematic PLDs and Include Files

```
[TITLE text]
[PATTERN text]
[REVISION text]
[AUTHOR text]
[COMPANY text]
[DATE text]
CHIP file_name device_name signal_list
[FASTCLOCK fastclock_name...]
[FASTINPUT input_name...]
[MINIMIZE {OFF | ON} [signal_name...]]
[NODE ({UIM | NODETRST}) signal_name...]
[PARTITION {partition_name | FB | FFB} signal_name...]
[STRING string_name  'string_text']

where: signal_list = {[/]signal_name | NC | VCC | GND}...
Note: FASTINPUT - Used only in PLFFB9 equation files
Note: MINIMIZE - Used in PAL files only.
Note: PARTITION - In PAL files is used for arithmetic only; not
                  applicable to PLFB9 or PLFFB9
```

Declaration Statements Used in Top-Level Files

```
[TITLE text]
[PATTERN text]
[REVISION text]
[AUTHOR text]
[COMPANY text]
[DATE text]
CHIP file_name XEPLD
[CEPIN /ce_name...[PIN pin_number ...]]
[FASTCLOCK fastclock_name...[PIN pin_number ...]]
[FOEPIN foe_name...[PIN pin_number ...]]
[INCLUDE_EQN  'file_name.extension']
[INPUTPIN [({RCLK = fastclock_name [CE = ce_name] |
     LE = fastclock_name | FI})] [/] signal_name...
     [PIN pin_number ...]]
[IOPIN [([RCLK = fastclock_name [CE = ce_name] |
     LE = fastclock_name] [PINFBK] [FOE = foe_name]
     [NODETRST])] [/] signal_name...[PIN pin_number ...]]
[LOGIC_OPT {OFF | ON} [signal_name...]]
[MINIMIZE {OFF | ON} [signal_name...]]
[MRINPUT]
[NODE [({UIM | NODETRST})]  [/] signal_name...]
[OPTIONS OFF [REG_OPT] [CLOCK_OPT] [UIM_OPT] [FOE_OPT]]
[OUTPUTPIN [([FOE = foe_name] [NODETRST])]
     [/] signal_name...[PIN pin_number ...]]
[PARTITION {partition_name | FBn | FBn_m | FFB | FB}
     signal_name...]
[PWR {LOW | STD} [signal_name...]]
[STRING string_name  'string_text']
```

Equation Statements Used in High Density Function Blocks

```
EQUATIONS
[[/]signal_name [:]= p_term [+ p_term]...]
[[/]signal_name [:]= p_term [+ p_term]... :+: p_term
  [+ p_term]...]
[signal_name [:]= {VCC | GND}]
[[signal_name.Dl = {p_term [+ p_term]... | VCC | GND}]
  [signal_name.D2 = {p_term [+ p_term]... | VCC | GND}]
  signal_name [:]= signal_name.D1 alu_op signal_name.D2]
[signal_name.{ADD | ADDMODE} = VCC]
[signal_name.CLKF = {p_term | fastclock_name | GND}]
[signal_name.FBK = { p_term [+ p_term]... | VCC}]
[signal_name.PRLD = {VCC | GND}]
[signal_name.RSTF = p_term]
[signal_name.SETF = p_term]
[signal_name.SHIFT = { p_term [+ p_term]... | VCC}]
[[/]signal_name.T = p_term [+ p_term]...]
[signal_name.TRST = {p_term | VCC}]

where:
p_term = [/]signal_name [.PIN] [* [/]signal_name [.PIN]] …
alu_op = {XOR | XNOR | AND | NAND | OR | NOR | VCC | GND |
D1_ONLY | D2_ONLY | NOTD1 | NOTD2 | D1_AND_NOTD2 |
NOTD1_AND_D2 | D1_OR_NOTD2 | NOTD1_OR_D2}
```

Equation Statements Used in Fast Function Blocks

```
EQUATIONS
[[/]signal_name [:]= {p_term [ + p_term]... | VCC | GND}]
[[/]signal_name [:]= p_term [+ p_term]... :+: p_term
  [+ p_term]...]
[signal_name.RSTF = p_term]
[signal_name.SETF = p_term]
[signal_name.CLKF = fastclock_name]
[signal_name.EXPORT = p_term [+ p_term]...]
[[/]signal_name.T = p_term [+ p_term]...]

where:
p_term = [/]signal_name [.PIN | .FI] [* [/]signal_name
  [.PIN | .FI]] …
```

XEPLD Fitter Modules and Files

This chapter describes the XEPLD Fitter modules, which convert a schematic or behavioral design into a file you can use to program an EPLD device. The intermediate files that the Fitter uses and the reports that the Fitter generates are also described.

XEPLD Fitter Modules

Fitting the design is the intermediate step between design entry and device programming. The XEPLD Fitter accepts a schematic or behavioral design and converts it into a file that specifies how to fit the design to an XEPLD device. During this process, the Fitter uses and creates many files, such as the log file, the error file, and reports.

When you fit an XEPLD design, the Fitter command (FITNET for schematic entry, FITEQN for behavioral entry) invokes several software programs, each with its own function. Figure 5-1 shows how the Fitter modules process your design. This diagram also shows the files each module produces.

Messages appear on the screen naming each program as it is invoked. All error or warning messages are written to the *design_name*.err file. Status and other information from individual programs is displayed on the screen and written to the *design_name*.log file. After fitting, the error and log files contain the results of the programs.

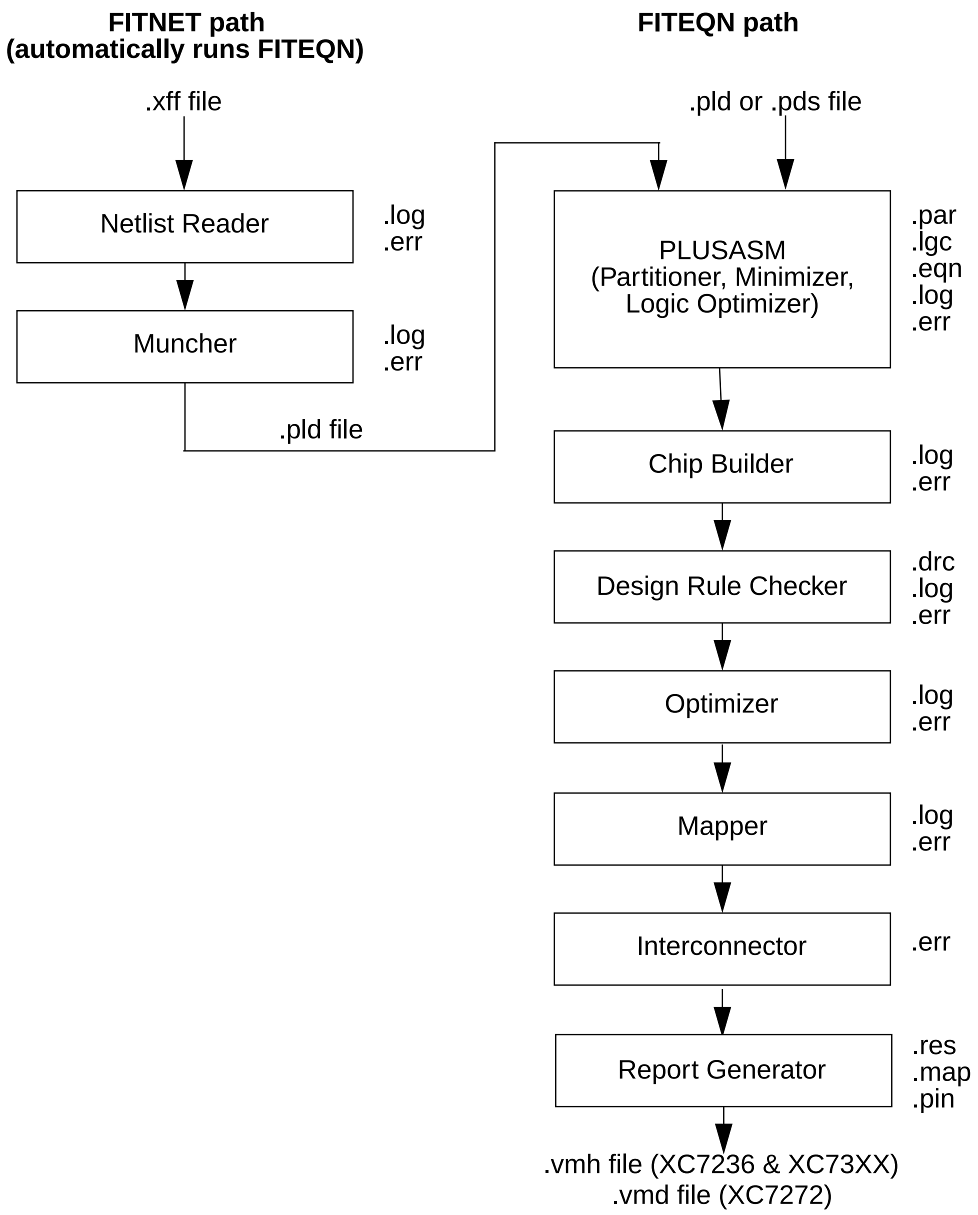


Figure 5-1 Fitter Flow of Modules and Files

The Netlist Reader (Schematic Only)

The FITNET command, which fits a schematic design, first calls the XNF netlist reader program to translate a netlist into an XEPLD design database file. The netlist reader reads the schematic netlist file and extracts all the XEPLD components and their interconnections and stores them in the design database. This design database is subsequently processed by other Fitter modules.

The netlist reader extracts only XEPLD 4.0 or XEPLD-compatible Unified Library 5.0 components. **You cannot mix these two types of components.**

The presence of non-XEPLD components causes errors.

To assign a signal to an EPLD pin, connect it to an input, output, or I/O buffer, then connect the buffer to a pad component.

The Muncher (Schematic Only)

The Muncher removes unnecessary logic from your design, thereby freeing up device resources. Sometimes your circuit design only needs part of a component (for example, only six bits of an eight bit component). The Muncher removes the unused portions of the component from your design. If an output of a component is not used, the Muncher first eliminates the output, then finds out if the inputs affecting this output affect any other used output. If any of these inputs does not affect any other used output, the Muncher eliminates the input. If an output from another component that was connected to the eliminated input does not drive another used input, then it becomes an unused output and is eliminated. The Muncher continues this process until no unused outputs remain.

If any input of a component is assigned a constant logic level (tied to VCC or GND in the schematic), the Muncher makes the necessary changes in the component logic and removes the input. The Resource report indicates which components have been affected by munching. If, for example, the load input of a shift register is permanently disabled, the Muncher eliminates the load control input, the parallel data inputs, and the load logic inside the shift register component.

The Logic Optimizer

The logic optimizer collapses the levels of logic to remove intermediate nodes. This improves the performance of the design.

The logic optimizer first removes all internal logic that is not used — that has no path to a primary output and is not explicitly in a PARTITION statement.

The logic optimizer moves logic forward by collapsing combinational expressions into expressions that reference them. If collapsing an expression into all referencing expressions succeeds, the expression becomes unused and is removed.

The logic optimizer does not collapse logic if the resulting logic uses more than the maximum available number of p-terms (shared + private) per macrocell, or more than the number of inputs available per Function Block.

The logic optimizer also moves forward any logic, whether combinational or sequential, that is buffered by a 3-state buffer, a clocked buffer, or a non-controlled buffer. However, logic that itself contains a 3-state or clock equation is not moved forward into a buffer that contains similar control equations.

You can prevent logic collapsing. In a behavioral design, use the LOGIC_OPT OFF statement. Refer to the “PLUSASM Command Reference” chapter of this manual for more information. In a schematic design, use the LOGIC_OPT=OFF attribute. See the *Interface User Guide* for your CAE tool for more information.

The logic optimizer also assigns FastCLK and FOE signals and performs input register assignment automatically.

FastCLK assignment attempts to move clock signals assigned to regular input pins to a FastCLK pin. This reduces the number of UIM inputs and product terms required by each Function Block and improves the performance of the design. Because the FastCLK pins do not connect to the UIM or to macrocell inputs, the software will not assign logic input signals to FastCLK pins. FastCLKs can only control macrocell and input register clocking.

You can turn off FastCLK optimization. In a behavioral design, use the OPTIONS OFF CLOCK_OPT statement. Refer to the “PLUSASM Command Reference” chapter of this manual for more information.

In a schematic design, use the CLOCK_OPT=OFF attribute. See the *Interface User Guide* for your CAE tool for more information.

Fast output enable (FOE) optimization moves a 3-state (OE) function assigned to a regular input pin to an FOE pin. Like FastCLK assignment, this reduces the number of UIM inputs and product terms required by each Function Block. Because the FOE pins do not connect to the UIM or to macrocell inputs, the software will not assign logic input signals to FOE pins. An FOE does not affect UIM feedback, so an output is not assigned to an FOE if it is declared using NODETRST.

You can turn off FOE optimization. In a behavioral design, use the OPTIONS OFF FOE_OPT statement. Refer to the "PLUSASM Command Reference" chapter of this manual for more information. In a schematic design, use the FOE_OPT=OFF attribute. See the *Interface User Guide* for your CAE tool for more information.

Input register optimization reduces the number of macrocells in a design by moving inputs that are registered by a macrocell register into a pad register, provided that no equation uses the non-registered version of the input. The clock by which the input register is controlled must be a FastCLK or an input that can be assigned to a FastCLK pin.

You can turn off input-pad register optimization. In a behavioral design, use the OPTIONS OFF REG_OPT statement. Refer to the "PLUSASM Command Reference" chapter of this manual for more information. In a schematic design, use the REG_OPT=OFF attribute. See the *Interface User Guide* for your CAE tool for more information.

You can turn off FOE, FastCLK, or input register optimization on a global level. You can turn off logic optimization globally or for individual outputs in a beharvioral design or components in a schematic design. Refer to the "PLUSASM Command Reference" chapter of this manual or see the *Interface User Guide* for your CAE tool for more information.

PLUSASM, the Partitioner, and the Minimizer

Associated with the PLUSASM assembler is the Partitioner/Minimizer module. This module is invoked automatically when you fit a behavioral design using FITEQN or a schematic design using XEMAKE. If you are assembling PLDs in a schematic design and using FITNET to fit the design, however, you must invoke PLUSASM as a separate command prior to fitting.

The PLUSASM assembler assembles equation source files. The Partitioner/Minimizer partitions a design with a large number of equations into groups that can fit into Function Blocks and performs Boolean minimization on the design equations.

The Partitioner makes all allocation decisions, including which AND gates go into the UIM. It subdivides equations that require more resources than are available in one EPLD Function Block. It intelligently distributes the logic functions among one or more Function Blocks. It places logic in the Fast Function Blocks first, then High-Density Function Blocks.

When filling each type of Function Block, the Partitioner first places the most resource intensive equation into the first partition, then tries to fit equations that share inputs and/or product terms into that first partition until the partition is full. Then it creates another partition, places the next largest equation, and repeats the process until all equations are placed into partitions.

When an equation requires more than the maximum available number of p-terms (shared + private) per macrocell, or more than the number of inputs available per Function Block, the Partitioner splits the equation and distributes intermediate equations among multiple macrocells. The subfunctions are then combined using the UIM-AND capability. The Partitioner may use a macrocell to combine split equations if the equations drive an external pin.

The PLUSASM assembler allocates the resources of each equation to the resources available in the Function Block of an EPLD. The types of resources it considers are inputs, private product terms, shared product terms, and outputs. If an equation has an equal or smaller number of product terms than the private p-terms available per macrocell, it automatically allocates them as private p-terms. If an equation requires more p-terms than the private product terms available per macrocell, it allocates the product terms according to frequency of use.

For detailed information about the resources available on each EPLD device, see the *Xilinx EPLD Data Book*.

In conjunction with the Partitioner, the Logic Minimizer minimizes the equations in each partition. The Minimizer takes into consideration product term sharing. By default, the Minimizer processes all equations except the ALU equations. You can optionally use the MINIMIZE=OFF command in the equation file to turn the Minimizer off for the entire design or for particular equations.

The Partitioner operation is transparent. PLUSASM invokes the Partitioner automatically for all PLDs except the PLPLD9 and PLPLD9F, which are defined in the image of a single High-Density or Fast Function Block. You can control the Partitioner for pin assignment and pin freezing operations with pin statements and PARTITION statements in the equation file. (In schematic designs, you use the LOC, F, and H attributes for pin assignment; see the *Interface User Guide* for your CAE tool.) The PLUSASM log report includes the Partitioner report, which shows the resulting partitions, minimization, and resource allocation for every partition and every equation.

You can view the final equations in the *design_name*.eqn file produced by the FITEQN command.

The Chip Builder

The chip builder loads into the XEPLD design database the architectural description of the selected target EPLD, defining all available logic and I/O resources. If you selected the -u option of FITEQN or FITNET, the Chip Builder drives unused I/O pads, clock signals, FOE signals, and CE pins to GND.

The Design Rule Checker

The architecture of any target EPLD cannot implement all the possible logic specifications, component configurations, and interconnections that may appear in a design. You must conform to some design rules to design with XEPLD. The Design Rule Checker (DRC) reads the design from the database and checks to see if any of the design rules have been violated. The following is a partial list of rules that are checked.

General Design Rule Violations

The DRC displays an error or warning if:

1. Open (hanging) inputs are found. Unless otherwise specified, all inputs of a library component must be connected or tied to VCC

or GND.

2. Some library components can only be used for a particular target EPLD. The DRC will generate an error if you attempt to use these components for other EPLDs. Restrictions on the use of components can be found in the library data sheets.

Pad Component Design Rule Violations

Pad component correct usage and applications are illustrated in the Library data sheets. The DRC displays an error if:

1. Two component outputs are connected to the same pad.
2. One component output is connected to two pads.
3. An input pad is connected directly to an output or I/O pad.
4. Pad pins are driven by VCC or GND.
5. Pad clocks are driven by VCC or GND.
6. Multiple input buffers are connected to the same pad (the exception is when an IBUF is used with an IFD, IFDX1, or ILD to receive a FastInput signal).
7. A pad is connected to a component other than an I/O buffer, or to another pad.
8. An IPAD is connected to an OBUF-type component.
9. An OPAD is connected to an input or control-input buffer (such as IBUF, BUFG, or IFD).

FastCLK, Clock Enable, and Fast Output Enable Violations

The DRC displays an error if:

1. There are more FastCLK, CE, or FOE pins in the design than the amount the target EPLD can support.
2. A FastCLK, CE, or FOE signal drives a component pin that is not a clock, CE, or FOE input.
3. A combination of fast clocks for logic components and I/O pads cannot be supported by the target EPLD.
4. The clocking requirement of a component is not met. Some com-

ponent clock inputs can only be driven by a fast clock and others only by a logic clock. Component clocking requirements are listed in the Library data sheets.

The Gate Optimizer

The Gate Optimizer processes UIM functions (AND, NAND, and inverter) as instructed by the Partitioner. Behavioral designs have special syntax for specifying AND/NAND functions.

The Mapper

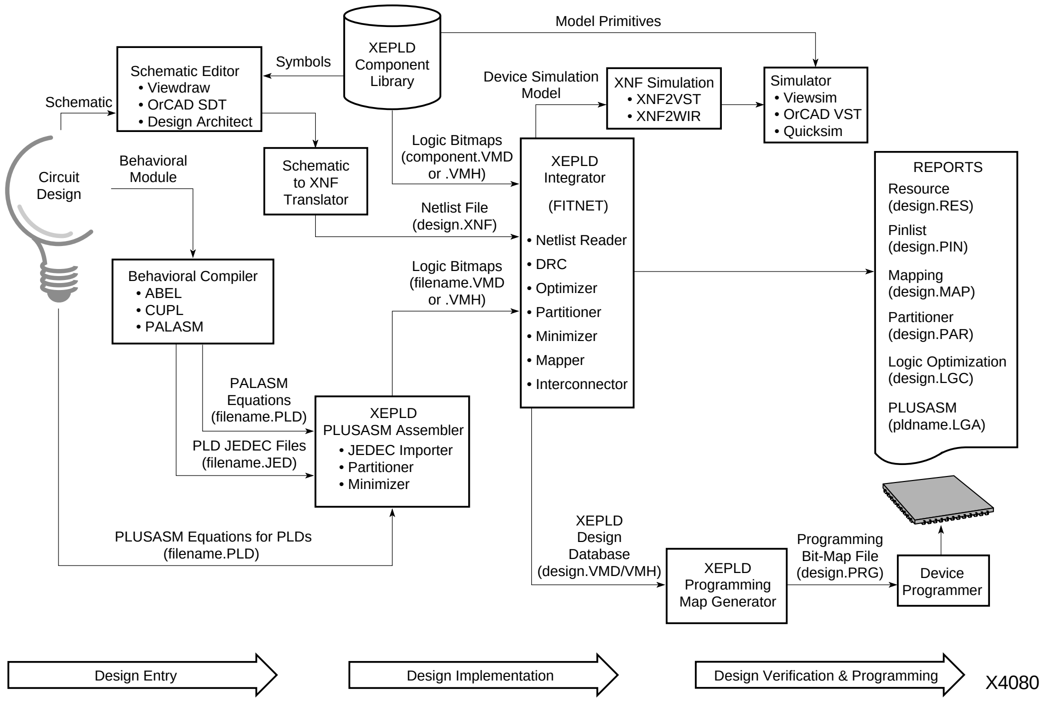
The Mapper module maps the logic of your design onto the architecture of the target EPLD as instructed by the Partitioner.

The Interconnector

The Interconnector connects the mapped components through the interconnect matrix (UIM). Because the UIM provides one interconnect path between each signal source (device input pin or macrocell output) and each Function Block input, it can always successfully interconnect everything that is placed on the EPLD. The Interconnector operation is therefore simple and fast.

XEPLD Files and Directories

The interrelationships of files in the XEPLD software are shown in Figure 5-2 and Figure 5-3. This section explains the files and directories that constitute the XEPLD Design System.



XEPLD Component Library
Model Primitives
Symbols
Schematic Editor • Viewdraw • OrCAD SDT • Design Architect
Schematic
Device Simulation Model
XNF Simulation • XNF2VST • XNF2WIR
Simulator • Viewsim • OrCAD VST • Quicksim
Circuit Design
Behavioral Module
Schematic to XNF Translator
Logic Bitmaps (component.VMD or .VMH)
XEPLD Integrator (FITNET) • Netlist Reader • DRC • Optimizer • Partitioner • Minimizer • Mapper • Interconnector
Netlist File (design.XNF)
REPORTS
Resource (design.RES)
Pinlist (design.PIN)
Mapping (design.MAP)
Partitioner (design.PAR)
Logic Optimization (design.LGC)
PLUSASM (pldname.LGA)
Behavioral Compiler • ABEL • CUPL • PALASM
Logic Bitmaps (filename.VMD or .VMH)
PALASM Equations (filename.PLD)
XEPLD PLUSASM Assembler • JEDEC Importer • Partitioner • Minimizer
PLD JEDEC Files (filename.JED)
PLUSASM Equations for PLDs (filename.PLD)
XEPLD Design Database (design.VMD/VMH)
XEPLD Programming Map Generator
Programming Bit-Map File (design.PRG)
Device Programmer
Design Entry
Design Implementation
Design Verification & Programming
X4080

Figure 5-2 Flow of Design Data Files Through FITNET

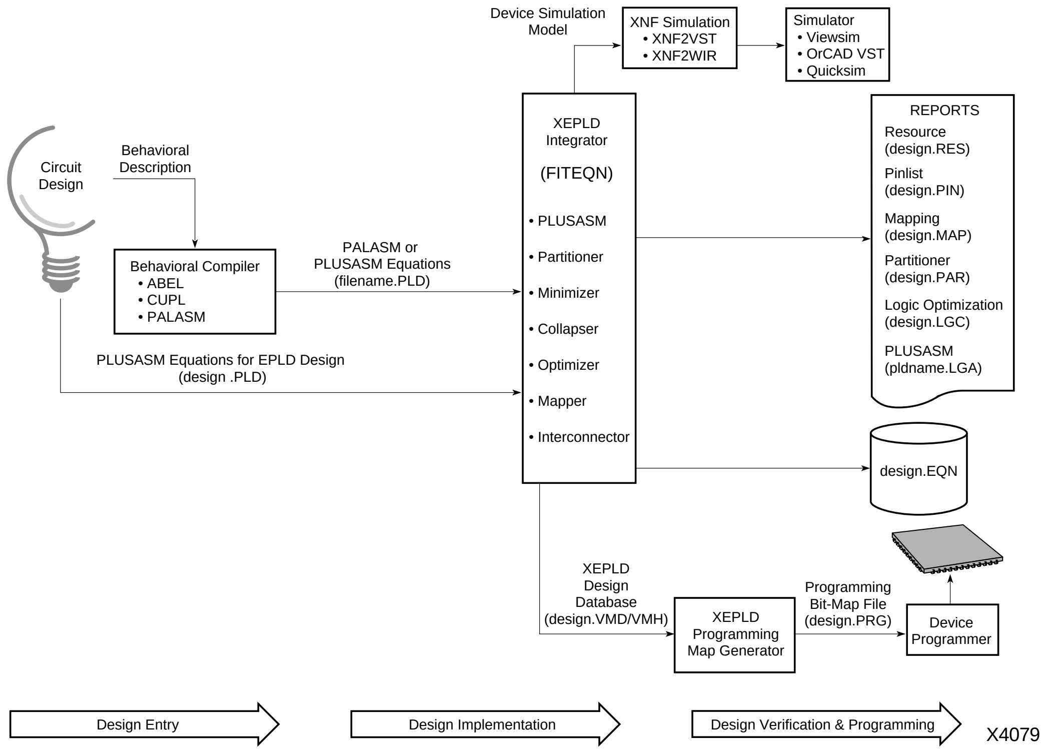

Device Simulation Model
XNF Simulation • XNF2VST • XNF2WIR
Simulator • Viewsim • OrCAD VST • Quicksim
Circuit Design
Behavioral Description
Behavioral Compiler • ABEL • CUPL • PALASM
PALASM or PLUSASM Equations (filename.PLD)
PLUSASM Equations for EPLD Design (design .PLD)
XEPLD Integrator (FITEQN)
• PLUSASM
• Partitioner
• Minimizer
• Collapser
• Optimizer
• Mapper
• Interconnector
REPORTS
Resource (design.RES)
Pinlist (design.PIN)
Mapping (design.MAP)
Partitioner (design.PAR)
Logic Optimization (design.LGC)
PLUSASM (pldname.LGA)
design.EQN
XEPLD Design Database (design.VMD/VMH)
XEPLD Programming Map Generator
Programming Bit-Map File (design.PRG)
Device Programmer
Design Entry
Design Implementation
Design Verification & Programming
X4079

Figure 5-3 Flow of Design Data Files Through FITEQN

On the PC, the XEPLD software is installed in the directory structure shown in Table 5-1.

Table 5-1 XEPLD Development Systems File Structure (PC)

DIRECTORY	FILES	
\XACT\ (or other path specified at installation time)		
		*.bat, *.exe
	DATA\	xepld.cfg
	DATA\CLIB\	*.vmd, *.vmh (no user-accessible files)
	TUTORIAL\	design_name.xff design_name.vmf *.pld, *.abl, *.pds (Users may run behavioral demos under this design directory)
	MSG\	*.txt (no user-accessible files)

On the workstation, the XEPLD software is installed in the directory structure shown in Table 5-1.

Table 5-2 XEPLD File Structure (Workstation)

DIRECTORY	FILES	
xact/bin/sparc or xact/bin/hppa (or other path specified at installation time)		
		*.bat, *.exe
	data/	xepld.cfg
	data/clib/	*.vmd, *.vmh (no user-accessible files)
	tutorial/	design_name.xff design_name.vmf *.pld, *.abl, *.pds (Users may run behavioral demos underthis design directory)
	msg/	*.txt (no user-accessible files)

The XACT root directory contains all the executables (.exe) and batch files (.bat). It also contains several subdirectories.

Subdirectory DATA contains the configuration file for the XEPLD executable (xepld.cfg)

Subdirectory DATA\CLIB contains the standard library component bit-maps (.vmd for the XC7272 and .vmh for all others).

The MSG subdirectory contains the XEPLD message files (.txt)

The TUTORIAL subdirectory is a design directory which contains sample behavioral designs and samples of XEPLD library component equation files.

The structure of the design directory and the file types it contains is described in Table 5-3.

Table 5-3 XEPLD User Design Directory File Structure

DIRECTORY	FILES	DESCRIPTION
design_directory\	design_name.xff	Merged and flattened netlist from XNFMERGE
	design_name.xnf	Netlist from CAE-tool or timing simulation netlist from VMH2XNF (XNF format)
	design_name.vmf	Pin-save file from XEPLD
	pld_name.pld	Equation source file for PLD component (in schematic) or for behavioral design (text), or equations derived from a schematic netlist
	design_name.int	PAL Interconnect report from XEPLD (text)
	pld_name.jed	JEDEC map source files defining PLD components
	design_name.jed	JEDEC formatted programming bit-map file from MAKEJED
	design_name.eqn	Shows the equations in your design after fitting (PLUSASM syntax)
	design_name.prg	HEX formatted programming bit-map file from MAKEPRG
	design_name.vmd or .vmh	XEPLD design database file from Fitter (binary)
	design_name.log	Execution log files from XEPLD modules (text)
	design_name.lga	Partitioner report for PLD equation file from PLUSASM (text)
	design_name.lgc	Logic Optimizer report (text)
	design_name.err	Error-log files from XEPLD modules (text)
	design_name.res	Resource report file from XEPLD (text)
	design_name.pin	Pinlist report file from XEPLD (text)
	design_name.map	Mapping report file from XEPLD (text)
	design_name.par	Partitioner report for complete behavioral design (text)
	design_name.sch	OrCAD/SDT source schematic file
	design_name.vsm	Simulation wirelist file from Workview (only for Viewsim modeling)
	design_name.vst	Simulation file for OrCAD VST
	design_name.inf	Netlist from OrCAD SDT
	design_name.als	Shows schematic logic functions, listing instance name, pin, net name, and component type
CLIB\	*.vmd, *.vmh	PLD bit-map files from PLUSASM assembler (binary)
SCH\	design_name.1	Workview source schematic file
WIR\	design_name.1	Wirelist file from Workview used as input to WIR2XNF

Input source files for XEPLD can be in the form of netlist files, equation files, JEDEC map files and pin-save files. Output data may take the form of a bit-map database, log files, report files, programming bit-map files, and simulation model files.

The Netlist File

The netlist file (*design_name*.xnf) is produced by SDT2XNF for OrCAD or WIR2XNF for Workview. Creating a netlist file is the first step in fitting your design to an EPLD device.

Mentor Graphics and CADENCE software can also produce .xnf files.

The PLUSASM Equation Files

Equation files (*pld_name*.pld or .pds, or *design_name*.pld) may be generated by an ASCII text editor, the JEDEC importer, Xilinx ABEL, or a commercial PLD compiler. The JEDEC importer translates JEDEC maps generated by a commercial PLD compiler into PLUSASM equation files and processes them through the PLUSASM assembler. The equation files are used to describe the functionality of a PLD in a schematic or the functionality of an entire behavioral design.

JEDEC Source Files

JEDEC source files (*pld_name*.jed) are used to define the functionality of PLDs in a schematic. These files are generated by commercial PLD compiler tools like ABEL, CUPL, etc.

Design Database File

This binary file (*design_name*.vmh or .vmd) is created during fitting and it contains a complete description of your fitted design. All the information for reports and export files are extracted from this database. The extension for this file is .vmd for XC7272 or .vmh for XC7236 and all XC7300-series devices.

The Report Files

Report files are generated automatically during fitting. They are ASCII files that describe the results of fitting. PLUSASM also generates a report when it assembles a PLD equation file or converts a PAL design. These report files have extensions .pin, .map, .res, .par, .lgc, .int, .eqn, and .lga for the Pinlist, Mapping, Resource, Partitioner, Logic Optimizer, PAL Interconnect, Equation, and PLUSASM Assembler reports, respectively.

The Log Files

The log files (*design_name*.log) are generated during fitting. They describe actions taken by the modules in terms of information messages, and error/warning messages.

The Error/Warning Files

Error files (*design_name*.err) are a subset of the log files and contain only the error and warning messages. The error and warning messages are displayed on the screen during the fitting process.

The Programming Bit-map File

This is an Intel-HEX or JEDEC formatted file (*design_name*.prg or .jed) that is generated after the fitting process at your request. It is used by a device programmer to configure a Xilinx EPLD device.

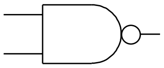
Appendix A

Typical Component Equations

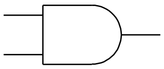
This appendix illustrates the behavioral equations that define the functionality of a variety of library components. You can use this information to better understand how to write behavioral equations or you can copy these equations into your design.

See Chapter 2 of the *XEPLD Design Guide* for more information on how to use the Xilinx PLUSASM language to create behavioral designs.

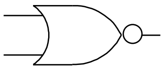
Basic Gates



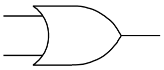
```
/o = i0*i1
```



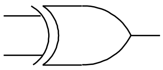
```
o = i0*i1
```



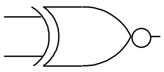
```
o = /i0*/i1
```



```
/o = /i0*/i1
```

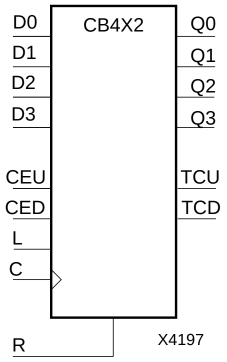


```
o = i0*/i1 + /i0*i1
```



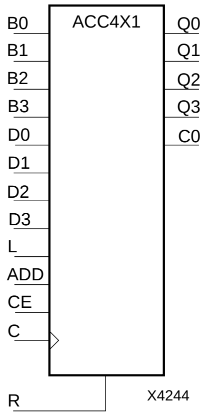
```
/o = i0*/i1 + /i0*i1
```

4-Bit Counter (CB4X2)



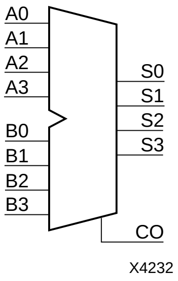
```
tcu       = ceu*q0*q1*q2*q3; UIM-AND
tcd       = ced*tcd3; UIM-AND
tcd3     := ceu*/l*q0*q1*q2*q3
          + ced*/ceu*/l*q0*/q1*/q2*/q3
          + l*/d0*/d1*/d2*/d3
          + /ceu*/ced*/l*/q0*/q1*/q2*/q3
          + r
tcd3.clkf= c
q3.d1     = ceu*/l*/r*q0*q1*q2
          + ced*/ceu*/l*/r*/q0*/q1*/q2
q3.d2     = l*/r*d3
q3.fbk    = /l*/r
q3       := q3.d1 xor q3.d2
q3.clkf   = c
q2.d1     = ceu*/l*/r*q0*q1
          + ced*/ceu*/l*/r*/q0*/q1
q2.d2     = l*/r*d2
q2.fbk    = /l*/r
q2       := q2.d1 xor q2.d2
q2.clkf   = c
q1.d1     = ceu*/l*/r*q0
          + ced*/ceu*/l*/r*/q0
q1.d2     = l*/r*d1
q1.fbk    = /l*/r
q1       := q1.d1 xor q1.d2
q1.clkf   = c
q0.d1     = ceu*/l*/r
          + ced*/ceu*/l*/r
q0.d2     = l*/r*d0
q0.fbk    = /l*/r
q0       := q0.d1 xor q0.d2
q0.clkf   = c
q0.prld   = gnd
q1.prld   = gnd
q2.prld   = gnd
q3.prld   = gnd
tcd3.prld= vcc
```

4-Bit Accumulator (ACC4X1)



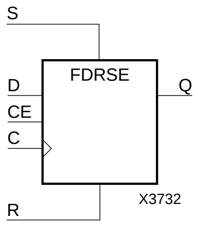
```
;cin generates carry into q0 when
;sutracting
cin.D1  = /add*ce*/l*/r
cin.D2  = /add*ce*/l*/r
cin     = cin.D1 gnd cin.D2
         ;cin macrocell output not used
q0.D1   = b0*add*ce*/l*/r
        + /b0*/add*ce*/l*/r
        + d0*l*/r
q0.fbk  = /l*/r
q0     := q0.D1 xor q0.D2
q0.add  = vcc
q0.clkf = c
q1.D1   = b1*add*ce*/l*/r
        + /b1*/add*ce*/l*/r
        + d1*l*/r
q1.fbk  = /l*/r
q1     := q1.D1 xor q1.D2
q1.add  = vcc
q1.clkf = c
q2.D1   = b2*add*ce*/l*/r
        + /b2*/add*ce*/l*/r
        + d2*l*/r
q2.fbk  = /l*/r
q2     := q2.D1 xor q2.D2
q2.add  = vcc
q2.clkf = c
q3.D1   = b3*add*ce*/l*/r
        + /b3*/add*ce*/l*/r
        + d3*l*/r
q3.fbk  = /l*/r
q3     := q3.D1 xor q3.D2
q3.add  = vcc
q3.clkf = c
```

4-Bit Adder (ADD4X1)



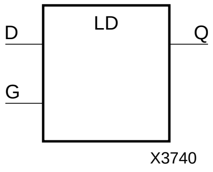
```
;Inputs
a0 a1 a2 a3 b0 b1 b2 b3

; a[3:0]          adder A-operand
; b[3:0]          adder B-operand

;Outputs
s0 s1 s2 s3

; s[3:0]          adder output
; co              carry-out from s3 via MC
carry chain
;                (co may only connect to ci
input of arith component or PLD)

PARTITION s3_0 s0 s1 s2 s3

EQUATIONS

s0.D1   = b0
s0.D2   = a0
s0     := s0.D1 xor s0.D2

s1.D1   = b1
s1.D2   = a1
s1     := s1.D1 xor s1.D2
s1.add  = vcc

s2.D1   = b2
s2.D2   = a2
s2     := s2.D1 xor s2.D2
s2.add  = vcc

s3.D1   = b3
s3.D2   = a3
s3     := s3.D1 xor s3.D2
s3.add  = vcc
```

Flip-Flop (FDSRE)



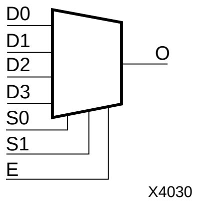
```
;Inputs
c d ce s r

;Outputs
q

EQUATIONS

q.d1   = d*ce*/r + s
q.fbk  = /ce*/r
q     := q.d1 or q.d2
q.clkf = c
```

Latch (LD)



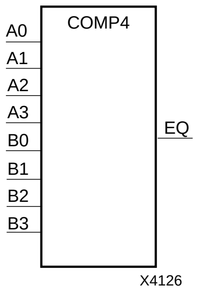
```
;Inputs
g d

;Outputs
q

EQUATIONS

q.rstf = /d*g
q.setf = d*g
q     := gnd
q.clkf = gnd
```

Multiplexer (M4_1E)



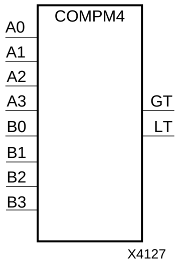
```
;Inputs
d0 d1 d2 d3 s0 s1 e

;Outputs
o

EQUATIONS

o = d0* /s1*/s0*e
  + d1* /s1* s0*e
  + d2*  s1*/s0*e
  + d3*  s1* s0*e
```

Comparator (COMP4)



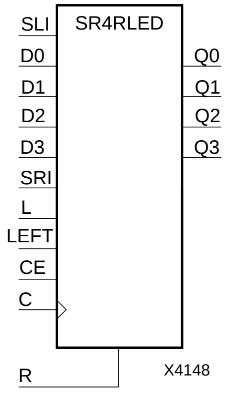
```
;Inputs
a0 a1 a2 a3
b0 b1 b2 b3

; a[3:0]        comparator A-operand
; b[3:0]        comparator B-operand

;Outputs
eq             ;A=B output

EQUATIONS

/eq     = a0*/b0 + /a0*b0
        + a1*/b1 + /a1*b1
        + a2*/b2 + /a2*b2
        + a3*/b3 + /a3*b3
```

Magnitude Comparator (COMPM4)



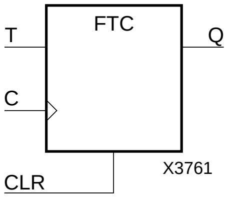
```
;Inputs
a0 a1 a2 a3 b0 b1 b2 b3

; a[3:0]          comparator A-operand
; b[3:0]          comparator B-operand

;Nodes
le2_0

; le[2:0]         intermediate A<=B terms

;Outputs
gt lt

; gt              A>B output
; lt              A<B output

PARTITION le2_0 gt lt

EQUATIONS

/le2_0    = a2*/b2
          + /a2*/b2*a1*/b1
          + a2*b2*a1*/b1
          + /a2*/b2*/a1*/b1*a0*/b0
          + /a2*/b2*a1*b1*a0*/b0
          + a2*b2*/a1*/b1*a0*/b0
          + a2*b2*a1*b1*a0*/b0

gt.D1     = /a3 + b3
gt.D2     = /a3*b3
gt.shift  = vcc
gt        = gt.D1 nand gt.D2
lt.D1     = /a0*b0
          + /a1*b1
          + /a2*b2
          + /a3*b3
lt.shift  = vcc
lt        = lt.D1 D1_AND_NOTD2 lt.D2
```

Shifter (SR4RLED)



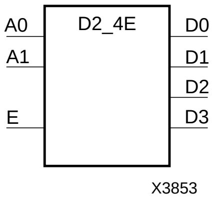
```
;Inputs
d0 d1 d2 d3 sli sri l left ce c r

; d[3:0]              parallel-load data
; sli                 shift-left serial input
; sri                 shift-right serial input
; l                   parallel-load enable
; left                shift direction: 1=left,
0=right
; ce                  shift clock enable
; c                   clock (optional FastCLK)
; r                   sync reset
;Outputs
q0 q1 q2 q3
; q[3:0]              counter output
EQUATIONS

q3       := left*ce*/l*/r*q2
          + /left*ce*/l*/r*sri
          + /ce*/l*/r*q3
          + l*/r*d3
q3.clkf   = c

q2       := left*ce*/l*/r*q1
          + /left*ce*/l*/r*q3
          + /ce*/l*/r*q2
          + l*/r*d2
q2.clkf   = c
q1       := left*ce*/l*/r*q0
          + /left*ce*/l*/r*q2
          + /ce*/l*/r*q1
          + l*/r*d1
q1.clkf   = c

q0       := left*ce*/l*/r*sli
          + /left*ce*/l*/r*q1
          + /ce*/l*/r*q0
          + l*/r*d0
q0.clkf   = c
```

T Flip-Flop (FTC)



```
;Inputs
c t clr

;Outputs
q

EQUATIONS

q.d1   = t
q.fbk  = VCC
q     := q.d1 XOR q.d2
q.clkf = c
q.rstf = clr
```

Decoder



```
;Inputs
a0 a1 e

;Outputs
d0 d1 d2 d3

EQUATIONS

d0  = /a1*/a0*e
d1  = /a1* a0*e
d2  =  a1*/a0*e
d3  =  a1* a0*e
```

PAL Devices Supported

This appendix explains how to include PALs of different types in Xilinx EPLD designs. It covers the following topics:

- Specific and generic PAL symbols for schematics
- PAL equation file syntax
- 22V10 and 20V8 support
- Generic PAL support
- PLFB9 and PLFFB9 support
- PALs supported through generic PAL components

Specific and Generic PAL Symbols for Schematics

The Xilinx schematic libraries contain these PAL symbols for EPLD devices:

- PL20V8 — The standard GAL20V8, with all features supported.
- PL22V10 — The standard 22V10 PAL, with all features supported.
- PLFB9 — Represents a XC7000 High-Density Function Block.
- PLFFB9 — Represents a Fast Function Block in an XC7000 device.
- PL20PIN — Generic 20-pin PAL symbol.
- PL24PIN — Generic 24-pin PAL symbol.
- PL48PIN — Generic 48-pin PAL symbol.

The last three PAL components can represent any PAL type (except 20V8 and 22V10) having the same number of pins or fewer. For example, the PL24PIN can represent PAL types such as 20RP6 or 18P4.

PAL Equation File Syntax

A PLUSASM equation file targeted at a PAL or used as an include file consists of three sections, as follows:

- Header (optional) — Gives the file a title. May also include information such as the author and creation date of the file.
- Declarations — Specifies the target PAL for the file in the CHIP statement and assigns signals to the PAL's pins in the pinlist.
- Equations — Defines the functions the PAL performs using boolean logic. Begins with the EQUATIONS keyword.

Here is an example equation file:

```
TITLE PAL1
CHIP   PAL1   P22V10;
;PINLIST (Highest pin number = 24)
     A B C NC NC NC NC NC NC NC NC NC NC NC NC NC
NC NC K J H G F NC
; PALCNVT Design Example PAL1
EQUATIONS
F := (B);
G := (F);
G.TRST = (C);
H := (G);
J := (B * K);
K := (B);
K.TRST = (C);
```

The CHIP statement has the following syntax:

CHIP *filename PALtype*

The pinlist follows the CHIP statement in the Declarations section. Every signal in the file must appear in the pinlist. Each signal corresponds to a PAL pin. The signals appear in order by pin number, beginning with the signal for pin 1. Pin 1 is the default clock for all registered equations in this file. For files used by schematic PAL components, use "NC", meaning no-connect, for unused pins. You can omit trailing NCs.

The same syntax applies to files included in behavioral designs and files that describe the logic of schematic PAL components.

Refer to the "PLUSASM Command Reference" chapter of this manual for more information about equation file syntax.

22V10 and 20V8 Support

PLUSASM provides special support for the features of these common devices, including the default clock, default 3-state control for registered outputs, and control of 22V10 synchronous set and asynchronous reset functions. These features are supported if any of the device types shown below appears in the CHIP statement of the PAL file.

Device	Device Type in Chip Statement
22V10	PL22V10, PAL22V10, GAL22V10, G22V10, P22V10, 22V10
20V8	PL20V8, PAL20V8, GAL20V8, G20V8, P20V8, P20V8R, 20V8

Default Clock

PLUSASM uses the clock signal (pin 1) as the default clock for each registered equation in a 22V10 or 20V8 equation file (or JEDEC file).

Default 20V8 3-State Control

PLUSASM uses the 3-state control signal (pin 13) as the default active-low 3-state control for each registered equation in a 20V8 equation file (or JEDEC file). For example, if pin 13 is "control_tri", the following 3-state control is implied unless otherwise specified:

```
registered_output_name.TRST = control_tri.
```

Global 22V10 Set/Reset

Since the 22V10 only has global set/reset functions, PLUSASM does not support individual .SETF and .RSTF control equations for the 22V10 equation files. Instead, PLUSASM supports the PALASM syntax for an implied 25th pin in 22V10 equations. This pin is used as a token output name for implementing a global synchronous set or asynchronous reset function. For example, the following control equations will synchronously set and asynchronously reset all 22V10 outputs:

```
25th_signal_name.SETF = p_term   ; emulate synchronous
set when set p_term is true
```

```
25th_signal_name.RSTF = p_term ; asynchronously reset
when reset p_term goes true
```

Automatic Inversion of Set and Reset

In the XEPLD architecture, all signal inversion is performed before the signal is an input to any register. This allows the set and reset functions of the register to operate as specified, without modification. In the 22V10, any signal inversion is performed after the register and therefore the set and reset functions of the register produce opposite results at the pin when the output is active low. PLUSASM automatically reverses the set and reset functions of any inverted 22V10 register in order to maintain the correct functionality when implemented in the EPLD.

Generic PAL Support

The generic PALs are the PL20PIN, PL24PIN, and PL48PIN library components. These require equation files with PLUSASM syntax. Your logic equations must be device-independent. However, in most cases, you can use your PALASM2 Boolean equation files generated by ABEL, CUPL, PALASM, and so on with no modifications.

If the CHIP statement of the PAL equation file contains a device type other than one of those listed under the “22V10 and 20V8 Support” section of this appendix, the PAL is considered generic. Tables A-1 to A-3 at the end of this appendix list the PALs for which the generic PALs can be substituted.

The XEPLD software doesn’t recognize the special features of any PALs except 20V8, 22V10, PLFB9, and PLFFB9, so all logic for a generic PAL must be device independent. There is one exception: unless otherwise specified, pin 1 is assumed to be the clock pin.

PAL types that have register set/reset capability, such as the 22RX8A, 16RA8, and 20RA10, require special attention if an active low output is being set/reset and the inversion takes place *after* the register. If this is the case, whenever the register is set, the inverted output signal goes low (and vice versa). In the XC7000 architecture, all inversions take place *before* the macrocell register. Whenever the register is set, the macrocell output always goes high, regardless of polarity. In this situation, you need to swap the register set and reset equations to maintain the same functionality in the XC7000 device.

For example, if this equation was in a 22RX8A file:

```
/x := y
x.setf = a
x.rstf = b
```

The equation would need to be changed to:

```
/x := y
x.setf = b
y.rstf = a
```

PLFB9 and PLFFB9 Support

These library components directly represent the XC7000 function blocks. You can target these library components with equation files that contain PLFB9 or PLFFB9 in the CHIP statements.

Clocks

The default clock for registered signals in the PLFB9 is pin 31. For the PLFFB9, it is pin 34. Pin 32 of a PLFB9 or pin 35 of a PLFFB9 can also be used as a FastCLK if you use the .CLKF dot extension. These pins are FastCLK only and cannot be used for other logic. For an example of how to use a FastCLK, refer to Chapter 4 of the *XEPLD Design Guide*.

PLFB9 Arithmetic Carry-In and Carry-Out Pins

These pins represent function block carry path connections, not UIM connections. Use them for arithmetic functions. Refer to Chapter 5 of the *XEPLD Design Guide* for examples.

Defining PLFFB9 Fast Inputs

Use the FASTINPUT statement to define which PLFFB9 inputs will be driven by the XC7000 FFB fast input pins.

PALs Supported through Generic PAL Components

The following tables list the PAL types that the generic PALs can represent with no modifications to their equation files.

Table B-1 PAL Types Directly Representable as PL20s

10L8	10H8	10P8	12L6	12H6	12P6
14L4	14H4	14P4	16L2	16H2	16P2
16R8	16RP8	16R6	16RP6	16R4	16RP4
16L8	16H8	16P8	16C1	16X4	18P8

Table B-2 PAL Types Directly Representable as PL24s

8L14A	6L16A	12L10	14L8	14H8	14P8
16L6	16H6	16P6	18L4	18H4	18P4
20L2	20H2	20P2	20R8	20RP8	20R6
20RP6	20R4	20RP4	20L8	20H8	20P8
20X4	20X8	20X10	20C1	20L10	20S10
20RS10	20RS8	20RS4	20RP10	20XRP4	20XRP6
20XRP8	20XRP10	22P10	22XP10		

Table B-3 PAL Types Directly Representable as PL48s

32R16

Equation Entry Application Note

Introduction

Welcome to the Xilinx EPLD equation entry tutorial. This tutorial will familiarize you with the behavioral entry methodology of the XEPLD development system. As a design example, we demonstrate the conversion of an existing UART receiver circuit, which is expressed entirely in Boolean equations using a PLD compiler and implemented using discrete PAL devices. The design was originally created in Xilinx ABEL as four separate PAL devices and was exported as .pld files (PLUSASM equations).

This tutorial will give you a basic understanding of the XEPLD development software and some hands-on experience so you can start your first behavioral design as quickly as possible. The same behavioral entry techniques apply to original behavioral designs as well as PAL design conversion.

See the "Converting PAL Designs" chapter in the *XEPLD Design Guide* for more information on PAL conversion techniques.

How to Follow this Tutorial

The tutorial consists of three sessions. Each session begins with an overview and summary of the interactive steps required.

You can finish this tutorial in about an hour. We supply all of the example equation files used in the design.

When you see an arrow between two commands, "→", it indicates that you select the first command and then select the second command from a menu that the first command displays. For example, **Fitter** → **FITEQN** means that you select **Fitter**, then select **FITEQN** from the menu that appears.

To end your XEPLD session at any time, select **Quit** from the XDM (Xilinx Design Manager) menu.

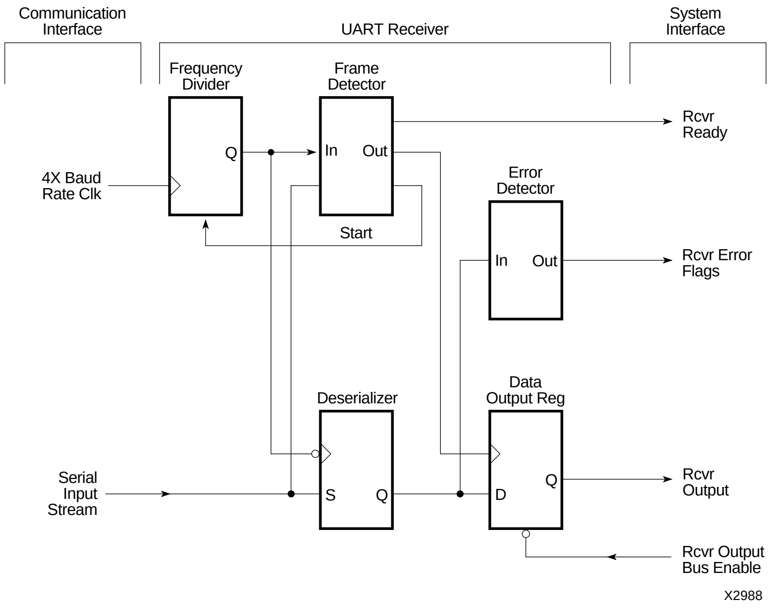


Figure C-1 UART Receiver Functional Logic Diagram

The Tutorial Design

In this tutorial, you will design the receiver section of a Universal Asynchronous Receiver Transmitter (UART). This circuit converts a serial data stream to parallel bytes and provides handshaking and error detection signals to the host system. Figure C-1 illustrates the functionality of this design.

The example design functions as follows:

1. A serial to parallel shift register (Deserializer) converts the serial stream to parallel data, which is latched to a register (Output Reg).
2. A simple state machine (Frame Detector) controls the receiver. Once the start bit is detected, the counter (Frequency Divider) begins to count sequentially, clocked by the 4X Baud Rate Clk.

3. The host is notified with the ready signal (Rcvr Ready) and reads the data by asserting the Rcvr Output Bus Enable signal.
4. The output of the counter is decoded to generate the control signals for the shift register, data latch, and error detection circuits. The following signals are generated:
   - Parity Error is generated if a byte parity is odd.
   - Framing Error is generated if any of the stop bits are low.
   - Overrun Error is generated if new data is ready to be latched into the output register before the CPU reads the previous data.

See Figure C-2 for the format of the serial input stream.

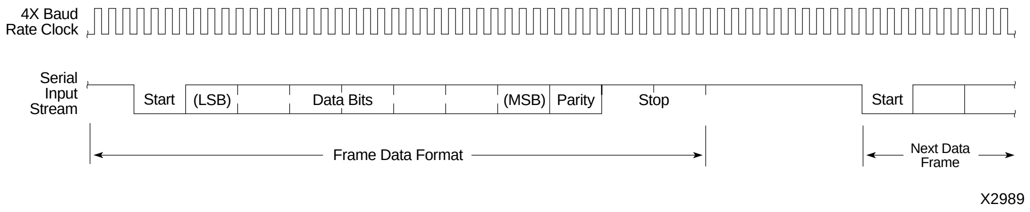


Figure C-2   UART Waveforms

Overview of the Sessions

The three sessions cover the following topics:

Session 1: Using the XEPLD Software

Session 2: Design Entry

Session 3: Fitting the Design

The Example Files

The design used in this tutorial is named UART (you will create the top-level file, uart.pld, as part of this tutorial). The Master Installer program installs all the files containing the uart example in a sample design directory named UART. The default path is "C:\XACT\TUTORIAL\BEHAVIOR\UART" on the PC and "*$XACT*/tutorial/behavior/uart" on the workstation. The file names are listed below.

Table C-1 Tutorial Files

ABEL Files	PLUSASM Files
rcvr.abl	rcvr.pld
cntr6.abl	cntr6.pld
shifter.abl	shifter.pld
datareg.abl	datareg.pld

An additional file, uartdemo.pld, is included to show a few edits you must make to the uart.pld file. During the course of this tutorial, you will use these sample files as examples of a completed design.

Session 1: Using the XEPLD Software

Session One concentrates on the XEPLD environment. The following tasks are explained in this session:

STEP 1: Prepare the System

STEP 2: Start XDM

STEP 3: Select Menu Items

STEP 4: Configure the XEPLD Environment

Step 1: Prepare the System

Before you start XDM, you must make sure that you installed all the software and that you set up the DOS environment. How you set up your environment depends on the platform on which you are running.

Setting Up the PC

To configure your PC for use with XDM, follow these steps:

1. Install the XEPLD development system. Refer to the installation instructions in the Release Notes for installation information.
2. Make sure the XACT software directory (containing XDM) is included in your PATH.
3. Make sure the XACT variable is set according to XACT installation requirements in your autoexec.bat file.
4. To specify a text editor to be used under XDM, set the EDITOR environment variable.

Setting Up the Workstation

To configure your workstation for use with XDM, follow these steps:

1. Install the XEPLD development system. Refer to the installation instructions in the Release Notes for installation information.
2. Make sure the XACT software directory (containing XDM) is included in the XACT environment variable in your .cshrc file. (If the $XACT variable contains only one directory, this also allows you to reference the XACT directory with $XACT. When $XACT is shown in italics, it means “substitute the directory pointed to by the XACT variable.”)
3. Copy the tutorial files from the $XACT path established during installation to a user directory, for example:

   ```
   % cd
   % mkdir xtutorial
   % cd xtutorial
   % cp $XACT/tutorial/behavior/uart/*.* .
   ```

4. To specify a text editor to be used under XDM, set the EDITOR environment variable in your .cshrc file.

Step 2: Start XDM

You can access all operations of XEPLD software from the XDM menu system.

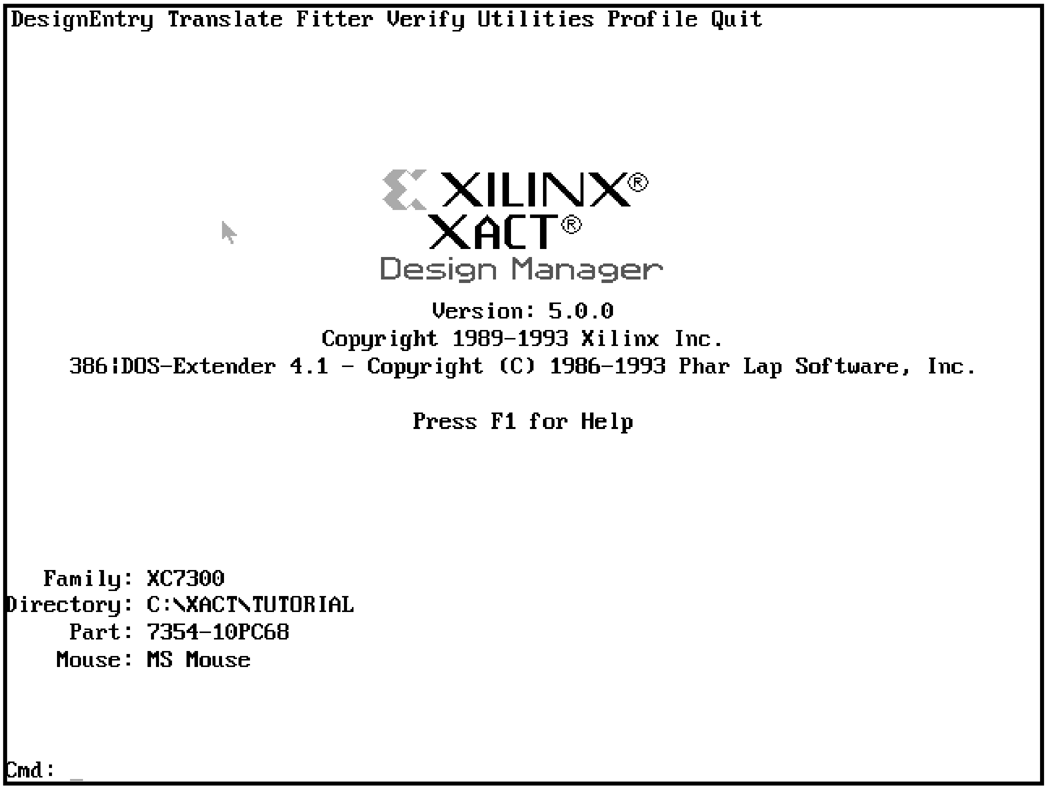
Invoke XDM by entering the following. If you are using a workstation, you must enter it is capital letters as shown.

```
> XDM
```

The XDM menu is displayed (Figure C-4 and Figure C-4).

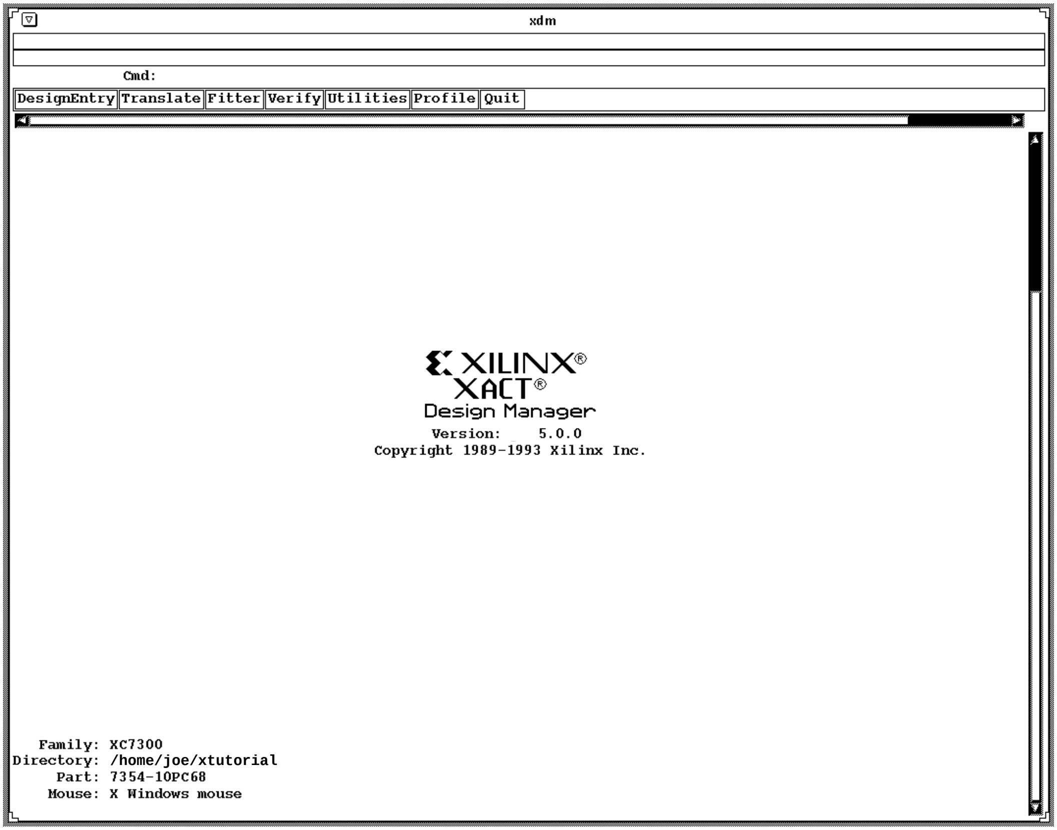
For detailed information regarding the menu items, refer to the "XACT Design Manager Menus" chapter of this manual.

Note: To end your XDM session any time, select **Quit** from the menu.



```
DesignEntry Translate Fitter Verify Utilities Profile Quit

                    XILINX®
                    XACT®
                Design Manager

               Version: 5.0.0
       Copyright 1989-1993 Xilinx Inc.
386|DOS-Extender 4.1 - Copyright (C) 1986-1993 Phar Lap Software, Inc.

              Press F1 for Help

   Family: XC7300
Directory: C:\XACT\TUTORIAL
     Part: 7354-10PC68
    Mouse: MS Mouse

Cmd: _
```

Figure C-3   The XDM Menu on the PC



```
Version:    5.0.0
Copyright 1989-1993 Xilinx Inc.
```

```
   Family: XC7300
Directory: /home/joe/xtutorial
     Part: 7354-10PC68
    Mouse: X Windows mouse
```

Figure C-4   The XDM Menu on the Workstation

Step 3: Select Menu Items

There are two methods for executing each of the commands available in the XDM menu:

- Use the mouse pointer to open the menu and click on the command you want. (You can also use the arrow and Enter keys in a similar manner.)
- Enter the command and any specified options by typing. What you type appears on the command line, which is the `Cmd:` prompt at the bottom of the screen on a PC or the top of the screen on a workstation.

Throughout this tutorial you will be instructed to select various menu items. You can use either the mouse or the command line to do so.

Using the graphic interface to execute commands or programs requires that you first know where that command resides in the XDM menu structure.

You also need to know which operations the mouse buttons are set to perform. The usage descriptions provided here assume the mouse is set to the default button configuration, as follows:

- Select for the left mouse button.
- Done for the middle mouse button.
- Menu for the right mouse button.

To open a menu, simply position the cursor on the menu you want to open and click the left mouse button.

On the PC, you can also cancel the last command selected by pointing anywhere except the menu titles and pressing the right button.

Notice that each command has a few letters highlighted (PC) or capitalized (workstation). The highlighted or capitalized letters indicate how you invoke commands at the command line using abbreviations.

Note: If you select a command on the PC and don't get the expected result, here are two things you can check:

- Some commands prompt you at the command line for more information, so you should always be aware of when the command line prompt changes. For example, when you select **Utilities → Directory**, the command line presents the `Directory:` prompt, and you must type a directory name if none of the menu choices are the directory you want.
- Other commands display a submenu with their own set of commands above the submenu. If you select an item on a submenu and nothing happens, select one of the commands above the menu, for example, **Done**.

Step 4: Configure the XEPLD Environment

How you configure your environment depends on which platform you are using.

Configuring the PC

To configure the PC, follow these steps:

1. First, the Directory (lower left corner) should be set to the tutorial directory path where the sample design files were installed.
2. Select the **Directory** field (or select **Utilities** → **Directory**) and enter the complete path to the tutorial directory at the command prompt, for example:

   ```
   Directory: c:\xact\tutorial\behavior\uart
   ```

   Select the **Done** command to redisplay the standard XDM menus.

   When you start up XDM, it always indicates the current directory.
3. Next, select **Family** to select the Xilinx device you want to use. For this tutorial, select **XC7300** to bring up a menu of the XC7300-series XEPLD devices. A submenu of parts appears. Select **7354PC68**. Next, a submenu of speeds appears. Select **10**.
4. You can customize the XDM interface by defining function keys using the Keydef command. For example, to invoke an external text editor with the push of a button, program one of the function keys using the **Profile** → **Keydef** command. If you want to use the DOS EDIT editor, for example, enter the following on the XDM command line:

   ```
   Cmd: keydef f2 dos edit\
   ```

   F2 is now programmed to invoke EDIT.

   The backslash (\) causes XDM to prompt you to finish the command when you press the defined key. You can use the backslash with any command that takes a variable argument at the end of its syntax. For example, if you define the F2 key as shown above and then press F2, you can answer the prompt with a file name as follows:

   ```
   Cmd: dos edit uart.pld
   ```

You use a text editor with XDM primarily for preparing Boolean equation (Behavioral entry) files for the PLUSASM equation assembler, and for viewing the reports generated by XEPLD.

Note: PLUSASM requires that text files contain only ASCII characters. If you use a word processor, it must be able to save files in ASCII (plain text) format.

Note: Do not change the file extensions that XDM assigns. XDM will not recognize or process files with incorrect extensions.

Configuring the Workstation

To configure the workstation, follow these steps:

1. First, the Directory (lower left corner) should be set to the tutorial directory path containing the sample design files.
2. Select the **Directory** field (or select **Utilities → Directory**) and enter the complete path to the tutorial directory at the command prompt. (You created this directory in the Setting Up the Workstation section of Session 1.) For example:

   `Directory:` *your_home_dir*`/xtutorial`

   Select the **Done** command to redisplay the standard XDM menus.

   When you start up XDM, it always indicates the current directory.
3. Next, select **Family** to select the Xilinx device you want to use. For this tutorial, select **XC7300** to bring up a menu of the XC7300-series XEPLD devices. A submenu of parts appears. Select **7354PC68**. Next, a submenu of speeds appears. Select **10**.
4. As an option, you can choose the editor that the Edit command brings up by setting the EDITOR environment variable in your .cshrc file, for example:

   ```
   setenv EDITOR vi
   ```

   You use a text editor with XDM primarily for preparing Boolean equation (Behavioral entry) files for the PLUSASM equation assembler, and for viewing the reports generated by XEPLD.

   On a workstation, setting up the Edit command is not really necessary because you can open another window for editing.

Note: PLUSASM requires that text files contain only ASCII charac-

ters. If you use a word processor, it must be able to save files in ASCII (plain text) format.

Note: Do not change the file extensions that XDM assigns. XDM will not recognize or process files with incorrect extensions.

Session 2: Design Entry

You can define the functionality of a design, or a portion of a design, either directly in the PLUSASM Boolean equation language, or through a third-party PLD compiler (for example, ABEL). The original files for this example are in ABEL syntax. These .abl files were compiled by Xilinx-ABEL to produce the .pld files.

This session demonstrates the development of the four PLD equation files that make up the uart design. You will learn how to break a design into smaller, manageable segments, develop each segment separately, and then consolidate the entire design.

The following tasks are explained in this session:

STEP 1: Segment the Design

STEP 2: Create Boolean Equations for each design segment

STEP 3: Consolidate the Design

STEP 4: Assign signals to specific EPLD pins (optional)

Step 1: Segment the Design

The first step is to break the design into small, manageable segments. Starting with a block diagram of the uart design (Figure C-1), you can easily identify dedicated functions (Table C-2).

Table C-2 Equation Files Used in the UART Design

Name	Description
SHIFTER	8-bit shift register, serial in parallel out
CNTR6	6-bit control counter
DATAREG	8-bit data latch with tristate outputs
RCVR	Random logic for control and error detection

The uart design is expressed as if it were first implemented using four discrete PAL devices. The equation files used to define the functionalities of these four PALs can be used without modification as input to XEPLD.

Step 2: Create Boolean Equations

Select **Utilities** → **Browse** or **Utilities** → **Edit**. A list of all the files in the design directory is listed. Select rcvr.pld from the list.

The equation file rcvr.pld is displayed in the browser as shown in Figure C-5.

```
TITLE RCVR

CHIP   RCVR   P22V10;

;PINLIST (Highest pin number = 24)
      x4clk c0 c1 c2 c3 c4 c5 read sdin d0 NC NC NC NC start bitclk
      byteclk par framing parity overun ready NC NC

;Control Logic and Error Detector for UART Receiver Design
;       Xilinx EPLD Applications, Feb. 93



EQUATIONS

/start := (/start * sdin
     + c0 * /c1 * /c2 * c3 * /c4 * c5);

start.CLKF = (x4clk);

bitclk := (start * /c0 * /c1);

bitclk.CLKF = (x4clk);

ready := (c0 * /c1 * /c2 * c3 * /c4 * c5 * /parity * /framing * /overun
     + ready * /read);

ready.CLKF = (x4clk);
```

Figure C-5   The rcvr.pld File Contents

```
byteclk := (/c0 * c1 * /c2 * /c3 * /c4 * c5 * /ready);

byteclk.CLKF = (x4clk);

overun := (/c0 * c1 * /c2 * /c3 * /c4 * c5 * ready
     + overun * /read);

overun.CLKF = (x4clk);

par := (start * /sdin * par
     + start * sdin * /par * bitclk
     + start * par * /bitclk);

par.CLKF = (x4clk);

parity := (parity * /read
     + /c0 * c1 * /c2 * /c3 * /c4 * c5 * par);

parity.CLKF = (x4clk);

framing := (/sdin * /c0 * /c1 * /c2 * c3 * /c4 * c5
     + framing * /read
     + /c0 * /c1 * /c2 * c3 * /c4 * c5 * /d0);

framing.CLKF = (x4clk);
```

Figure C-5 The rcvr.pld File Contents (continued)

The rcvr.pld file represents a portion of the entire uart design. Its equations are merged with those of the three other PLD files.

The keyword EQUATIONS identifies the beginning of the equation section, which contains Boolean equations for each output signal used. Take some time to examine the equations.

Exit from Browse (or Edit). If you wish, examine the remaining equation files shifter.pld, datareg.pld, and cntr6.pld.

You can also examine the corresponding source .abl files. rcvr.abl is shown in Figure C-6.

```
module rcvr
title 'Control Logic and Error Detector for UART Receiver Design
       Xilinx EPLD Applications, Feb. 93'

        rcvr  device  'p22v10';

" Inputs
  x4clk              pin 1;           " External clock (4x baud rate)
  c0,c1,c2,c3,c4,c5  pin 2,3,4,5,6,7; " State counter outputs (from cntr6)
  read               pin 8;           " Read enable (from cntr6,active-high)
  sdin               pin 9;           " Serial data input (external)
  d0                 pin 10;          " Shift register LSB output

" Outputs
  start              pin 15 istype 'reg'; " Start bit detector
  bitclk             pin 16 istype 'reg'; " Bit clock (to shifter)
  byteclk            pin 17 istype 'reg'; " Output data register clock
  par                pin 18 istype 'reg'; " Parity accumulator
  framing            pin 19 istype 'reg'; " Framing error output (external)
  parity             pin 20 istype 'reg'; " Parity error output (external)
  overun             pin 21 istype 'reg'; " Overrun error output (external)
  ready              pin 22 istype 'reg'; " Receiver ready output (external)

" Variables
  count = [c5..c0]; " c5 is MSB

Equations

!start := !start & sdin           " Start goes high when sdin goes low;
        # (count == 41);          " start stays high until count=41.
start.clk = x4clk;

bitclk := !c0 & !c1 & start;      " Bitclk pulses every 4 cycles.
bitclk.clk = x4clk;

ready := (count == 41) & !parity & !framing & !overun
       # ready & !read;           " Ready goes high at count=41 if no errors
ready.clk = x4clk;                " and stays high until register read.

byteclk := (count == 34) & !ready; " Strobe data register at count=34
byteclk.clk = x4clk;               " only if ready not still active.

overun := (count == 34) & ready  " Overrun error at count=34 if ready still on;
       # overun & !read;         " overun stays on until register read.
overun.clk = x4clk;
```

Figure C-6 The rcvr.abl File Contents

```
par := (par $ sdin) & bitclk & start
     # par & !bitclk & start;       " Accumulate parity of sdin on each bitclk;
par.clk = x4clk;                    " reset while start=0.

parity := (count == 34) & par     " Parity error at count=34 if par odd (1);
      # parity & !read;           " parity stays on until register read.
parity.clk = x4clk;

framing := (count == 40) & (!sdin # !d0) " Framing error at count=40 if either
      # framing & !read;                 " stop bit low;
framing.clk = x4clk;                    " framing stays on until register read.

end
```

Figure C-6   The rcvr.abl File Contents (continued)

Step 3: Consolidate the Design

To consolidate all the segments of the design and define the EPLD device I/O, use the PALCONVT command. This command allows you to automatically convert a design made up of individual PAL files.

These PAL files can be PLUSASM files that you created using a text editor, ABEL-generated PALASM-2 boolean equation files (or files from some other third-party design entry package), or JEDEC files converted with the JED2PLD command.

1. Select the **Fitter** → **PALCONVT** command.
2. When prompted for a design file, choose a name for the new converted PAL design in response to the command prompt:

   ```
   Enter design file name (.pld): uart
   ```

3. A list appears containing all the .pld and .pds files in the current directory. Select the following files from the list:

   rcvr.pld
   cntr6.pld
   shifter.pld
   datareg.pld

4. Select **Done** to complete file selection.
5. When prompted for a target, select **Create new PLD and PAL Interconnect Report**.

   You can use PALCONVT and FITEQN in one step, but it is a good idea to perform the steps separately the first time you fit a design, because you may want to edit the top-level file that PALCONVT creates.

   The PALCONVT command reads the PAL file equations, resolves any polarity inversions, resolves any PAL-specific functionality, and automatically determines all external requirements for dangling signals.
6. Examine and edit the uart.pld file by selecting **Utilities** → **Edit**.

   Figure C-8 shows the resulting main equation file.

.

```
PATTERN uart.pld  - file made by PALCNVT command

CHIP  uart  XEPLD

INCLUDE_EQN 'shifter.pld'
INCLUDE_EQN 'rcvr.pld'
INCLUDE_EQN 'datareg.pld'
INCLUDE_EQN 'cntr6.pld'

INPUTPIN SDIN X4CLK RD CS
OUTPUTPIN  DOUT7 DOUT6 DOUT5 DOUT4 DOUT3 DOUT2 DOUT1 DOUT0

NODE BITCLK D7 D6 D5 D4 D3 D2 D1 D0 C0 C1 C2 C3 C4 C5 READ START
                   BYTECLK PAR FRAMING PARITY OVERUN READY

EQUATIONS
```

Figure C-7   The uart.pld Main Equation File

The *design_name*.pld file that PALCONVT creates contains pin declarations, NODE declarations, and INCLUDE_EQN statements for the design. You can edit this file to specify additional external signals, for example under the OUTPUTPIN declaration.

It is also important to include .PRLD equations to ensure that all the registers in the chip are preset properly when the power is first turned on.

7. Note that the FRAMING, PARITY, OVERUN, and READY signals are interpreted as NODEs when they are actually OUTPUTPINs. Edit the uart.pld file to reclassify these signals. Although the BITCLK, BYTECLK, and START signals really are nodes, we recommend moving them to the OUTPUTPIN listing to prevent them from being optimized; this ensures that they are visible during timing simulation.

8. Add .PRLD equations for each output in the RCVR part of the design. You only need to preload these signals because these signals control the rest of the design.

   Figure C-8 shows the edited main equation file. The uartdemo.pld file also includes these edits.

.

```
PATTERN uart.pld  - file made by PALCNVT command

CHIP  uart  XEPLD

INCLUDE_EQN 'shifter.pld'
INCLUDE_EQN 'rcvr.pld'
INCLUDE_EQN 'datareg.pld'
INCLUDE_EQN 'cntr6.pld'

INPUTPIN SDIN X4CLK RD CS
OUTPUTPIN  DOUT7 DOUT6 DOUT5 DOUT4 DOUT3 DOUT2 DOUT1 DOUT0

NODE  D7 D6 D5 D4 D3 D2 D1 D0 C0 C1 C2 C3 C4 C5 PAR READ

; declarations added after palconvt:

outputpin start byteclk bitclk framing parity overun ready

EQUATIONS

; equations added after palconvt:

START.PRLD   = GND;
BITCLK.PRLD  = GND;
READY.PRLD   = GND;
BYTECLK.PRLD = GND;
OVERUN.PRLD  = GND;
PAR.PRLD     = GND;
PARITY.PRLD  = GND;
FRAMING.PRLD = GND;
```

Figure C-8 The uart.pld Main Equation File after Editing

The declaration section contains design identification information plus device and pin data. The PLUSASM keywords TITLE, AUTHOR, COMPANY, and DATE identify the design. The keyword CHIP identifies the design name (uart) and device type (XEPLD), which identifies a behavioral design targeted to an EPLD device. The particular EPLD device is selected in the XDM menu. In this example, we selected the XC7354-10PC44 as our target device.

The INPUTPIN, FASTCLOCK, OUTPUTPIN, and NODE statements in the declarations section identify all signal names in the design and designate them as input, clock, output, and internal node names, respectively.

The file contains an INCLUDE_EQN statement for each of the four constituent equation files. The main file in this example contains no functional operations.

The PIN statement, which is part of the OUTPUTPIN statement in the uart.pld file, is used for pin assignment and is described in the next step.

The .PRLD equations are all listed under the EQUATIONS keyword. See the "PLUSASM Command Reference" chapter for more information about how to use PRLD signals.

The PALCONVT command generates a PAL Interconnect report named uart.int, which summarizes the number of equations found and I/O pins created. You can use this report to help you choose the best target device and to verify that the PALs are connected properly..

View the PAL interconnect report, uart.int. You can use the **Utilities** → **Browse** or **Utilities** → **Edit** command.

As an alternative, you could have entered all your equations for the uart design in the EQUATION section of a single PLUSASM file.

Step 4: Assign Signals to Specific EPLD Pins

As an option, you can assign signals to specific EPLD pins. To do pin assignment, you must be aware of the architecture of the target EPLD device so that you do not, for example, assign an output to an input pin.

By assigning a list of pin numbers to a series of signals in an INPUTPIN, OUTPUTPIN, or IOPIN statement, those outputs are automatically assigned to the corresponding set of device pins.

You can add the following PIN statement to the end of the OUTPUTPIN statement of the main uart equation file:

```
OUTPUTPIN  DOUT7 DOUT6 DOUT5 DOUT4 DOUT3 DOUT2 DOUT1
DOUT0
         FRAMING PARITY OVERUN READY BITCLK BYTECLK START
         PIN 18 19 20 21 22 24 25 26
```

This statement assigns signals DOUT0 to DOUT7 to specific pins of the targeted XC7354 EPLD device.

The remaining signals in the equation file are allocated automatically by the XEPLD software.

Session 3: Fitting the Design

Session 3 concentrates on the XEPLD Fitter. The following tasks are explained in this session:

STEP 1: Invoke the Fitter

STEP 2: View the Reports

STEP 3: Save Pin Assignments

STEP 4: Create the Programming File

STEP 5: Create a Simulation Model (optional)

Note: It is at this point that the Family, Part, and Speed values you selected in Session One are used. The XC7354 device was chosen for this tutorial because the uart design is likely to fit into it.

Review the PAL Interconnect report (uart.int), the Resource report (uart.res), and if necessary the Partitioner report in the log file (uart.par) after fitting to determine the actual resource utilization of your design.

Step 1: Invoke the Fitter

To invoke the Fitter, follow these steps:

1. Select **Fitter → FITEQN**.
2. When a list of options is displayed, do not choose an option; just select **Done**.
3. Select uart.pld from the list of file names displayed.
4. Select **New File** when prompted for an output file, then type **`uart`** at the command line prompt.

If you have saved pin allocation information from a prior fitting in a .vmf file (using the PINSAVE command), the FITEQN command allows you to use this file to preserve your prior pinout; refer to Step 3 below.

The XEPLD Fitter is composed of several sub-modules. As the Fitter processing proceeds, a message is displayed on the screen indicating which sub-module is running. The FITEQN modules produce a database file (uart.vmh). From this database, a programming file can be produced to program the device. Simulation models can also be produced from this database file.

If the Fitter encounters errors, it displays them on the screen and stores them in a file (uart.err) for future reference. If errors are encountered, you can press Ctrl-C at any time to stop the execution and look at the error and warning logs.

Step 2: View the Reports

The following extensions designate the reports produced by the FITEQN command:

.res	The Resource report
.map	The Mapping report
.pin	The Pinlist report
.eqn	The Equation file
.par	The Partitioner report

You can view the reports that the Fitter generates using the **Utilities → Browse** or **Utilities → Edit** command.

1. First view the Resource report, uart.res.

   The Resource report lists the amount of resources that were used to implement the design. This report contains the total number of Function Blocks used and input/output (I/O) pins used on the target device. These totals are subtracted from the total resources of the device to give the amount of remaining resources available to the designer. This report also lists any portions of the design that were not mapped due to space limitations or design errors.

2. Next, view the Mapping report, uart.map.

   The Mapping report lists each Function Block in the device and details which output signals were mapped to that Function Block and how they were mapped. The Mapping report is used primarily for design debugging and to assist manual mapping.

3. Next, view the Pinlist report, uart.pin.

   The Pinlist report provides the designer with chip pin placement information. For each pin on the package, the Pinlist report indicates the operation of the pin as used in the design and the signal from the design appearing on the pin.

4. Next, view the Equation file, uart.eqn.

The Equation file shows how the equations that describe your design were optimized and mapped onto the EPLD device.

5. Finally, view the Partitioner report, uart.par.

   The Partitioner report shows the allocation of Function Block resources and lists all the equations in the design. This report helps you identify and correct design errors and provides information for optimizing or modifying your design.

Step 3: Save Pin Assignments

Use the **Translate → PINSAVE** command after a successful fitting of your design to save the pin allocation information into a Pinsave file, uart.vmf.

This allows you to preserve the saved pinout in case you need to make a design change or add more logic.

A message similar to the following is displayed on your screen:

```
Writing pin allocation in C:\XDM\TUTORIAL\UART.VMF
```

If you set the -f (Pin-freezing on) option of the FITEQN command to On, the Fitter assigns the pins to the locations indicated in the Pinsave file. This allows you to assign pins to the same positions with each iteration of your design. The -f option is Off by default. Selecting the -f option repeatedly before you select Done toggles the -f option On and Off. The On or Off setting of this option is displayed in a status line at the bottom of the XDM screen above the command line.

Step 4: Create the Programming File

After you have fitted your design, create a programming file by selecting **Verify → MAKEPRG**. Type your initials when prompted for a signature. Select uart.vmh from the list of file names displayed.

If you installed the Xilinx DS120 programmer, PROLINK appears under the Verify menu. This is the DS120 programmer control and interface software that would be used to download the uart.prg file to the programmer. Refer to the DS120 documentation for instructions.

You can also create a JEDEC programming file, required by other third-party programmers, by using the **Verify → MAKEJED** command and selecting uart.vmh.

Step 5: Create a Simulation Model

As an option, you can create a model of the completed design for OrCAD or Viewlogic timing simulation.

After fitting your design, you can create an OrCAD VST or Viewsim wirelister file for timing simulation using a single command.

1. Select **Verify** → **XSIMMAKE**.
2. Select **OrCAD_Epld_Timing** or **Viewlogic_Epld_Timing** as the target.
3. Select **uart.vmh** from the list displayed.

XSIMMAKE invokes the following XDM commands, which you can invoke yourself if you want to see each step in the translation process. You do not need to select any of the options on any of the commands.

1. Use the **Verify** → **VMH2XNF** command in the XDM menu and select uart.vmh from the file list. This creates a new .xnf file that contains an image of the EPLD device and its timing parameters.
2. Use the **Verify** → **XNF2VST** or **XNF2WIR** command in the XDM menu and select uart.xnf from the file list. This creates a model, expressed as an OrCAD VST or Workview WIR file, of an EPLD device containing the UART design.
3. Use the **Verify** → **VSM** command (Viewlogic only). Select uart.1 from the list of files. This creates a Viewsim wirelister file (uart.vsm) for functional and timing simulation.

Glossary

This appendix provides definitions of the words and terms used throughout the *XEPLD Design Guide* and the *XEPLD Reference Guide*.

ABEL — A high-level PLD design compiler, available from DATA I/O corporation.

Arithmetic Equations — Equations that specify the special arithmetic capabilities of the Xilinx EPLDs.

Behavioral Design Method — Defines a circuit in terms of a textual language rather than a schematic of interconnected symbols.

CAE Tool — (Computer Aided Engineering Tool) Usually refers to Viewlogic (ViewCad), OrCAD (OrCAD PLD), or Mentor.

CUPL — A PLD development tool available from Logical Devices Inc.

EPLD — Erasable Programmable Logic Device.

Equation Splitting — An automatic process performed by XEPLD to divide large behavioral equations into smaller functions that will fit within the available device macrocell resources.

Fast Carry — Arithmetic carry functions using the dedicated fast carry chain that interconnects macrocells. These signals do not pass through the UIM.

Fast Function Block (FFB) — Provides fast pin-to-pin logic throughput for critical decoding and ultra-fast state machine applications (XC7300 family only). The output pins associated with Fast Function Blocks have high current drive capability.

Fast Output Enable (FOE) — Tristate control signals that use the dedicated FOE wiring of the device, and not the UIM wiring.

FastCLK — A clock signal that uses the dedicated FastCLK wiring of the device, and not the UIM.

FastInput — Inputs to the device that connect directly to the function block inputs, bypassing the UIM.

Function Block — The High Density Function Blocks of the device, designed to provide the maximum logic density, containing nine macrocells. The output pins associated with Function Blocks have the standard current drive capability.

function block — Either a High Density Function Block or a Fast Function Block.

Fitter — The software that maps a PLD logic description into the target device.

Generic PAL — Any PAL device type other than 22V10 or 20V8.

I/O Blocks — The input/output logic of the device containing pin drivers, registers and latches, and 3-state control functions.

Include File — Equation files that are specified by an INCLUDE_EQN statement in a Top-Level File.

Input Pad Registers and Latches — D Type registers located in the I/O pad sections of the device. Input pad registers can be used instead of macrocell resources.

JEDEC — A file format used for downloading device bitmap information to a device programmer.

Linked Equations — Any equation that uses either the .ADD, .ADDMODE, .EXPORT, or .SHIFT extensions. (These equations must be physically adjacent in the device.)

LOG/iC — A PLD compiler, available from ISDATA.

Macrocell — The basic unit of logic in the device. A macrocell can implement both combinational and registered equations. High Density Function Block macrocells also contain an ALU for implementing arithmetic functions.

Node — Any signal used only as feedback.

Optimization — The process of reducing your design to the minimal required device resources. Optimization includes collapsing of combinational logic nodes into device outputs and registers, assigning signals to global FASTCLOCK and FOE nets, utilization of I/O buffer registers, and the creation of UIM-AND functions.

OrCAD PLD — A PLD compiler, available from OrCAD.

PAL — Programmable Array Logic.

PALASM — A PLD compiler available from Advanced Micro Devices. The Xilinx PLUSASM language is based on PALASM and can accept most PALASM files.

Partitioning — The process of placing symbolic logic into the physical structures of the device. The basic partition is the Function Block or Fast Function Block.

Pin — The physical XC7000 device pins (external connections).

Pin Feedback — Specifies that the associated signal comes from the actual device pin and not from the UIM.

PLD — Programmable Logic Device.

PLUSASM — The Xilinx native behavioral design language for EPLD development.

Product Term Cascading — The process of passing product terms (in groups of four) from one macrocell to another for the purpose of increasing the number of usable product terms. (See the PLUSASM .EXPORT command.)

Top-Level File — The main file of a PLUSASM design. It contains design control information and either design equations or references to Include Files containing design equations.

Universal Interconnection Matrix (UIM) — The primary device resource used to interconnect macrocells. Propagation delays through the UIM are constant and independent of the interconnections. AND functions can also be implemented in the UIM.

UIM-AND Function — An AND gate created from the inherent wired-AND structure of the UIM; requires no macrocell resources.

UIM Feedback — Specifies that the associated signal comes from the macrocell and not from the device pin.

Wired-AND Functions — AND gates (and their DeMorgan equivalents) produced by the inherent structure of the UIM.

Xilinx ABEL — The Xilinx ABEL compiler.

XACT — The Xilinx design editor.

XEPLD — Xilinx EPLD development software.

XDM — The Xilinx XACT Design Manager, the user interface to XEPLD.

Index

A

ABEL, 5-15
    example files, C-11
    see also PLD compilers
Abel file
    translating, 2-20
ABL2PLD command, 2-20
active window
    in XDM, 2-6
ADD equation, 4-44
ADDMODE equation, 4-46
AND gates,optimization, 5-9
ANNOTATE command, 2-20
Apollo, 2-4
Architecture
    and design constraints, 5-7
    mapping design to, 5-9
Arithmetic
    carry enable equation
        ADD, 4-44
        ADDMODE, 4-46
    carry-in, 4-44
Arrow keys
    activating, 2-38
ASCTOVST command, 2-29
Assembling an equation file, 2-22
Asynchronous reset equation (RSTF), 4-59
Asynchronous set equation (SETF), 4-60
AUTOEXEC.BAT, PATH statement for XDM, 2-2, 2-4

B

BAT files, 5-13
Behavioral design, C-11
    integrating, 2-24
    overview, 1-9, 1-10
Bit-maps, in library, 5-13
Browse command, 2-34, 3-1
Button settings on mouse, C-8

C

CE
    design rules, 5-8
CEPIN statement, 4-5, 4-6
Checker, for design rule violations, 5-7
Chip builder integrator module, 5-7
CLEANUP command, 2-20
CLIB directory, 5-13
CLKF equation, 4-48
Clock Enable see CE
Colors of menu
    changing arrangement, 2-39
    changing palette, 2-40
Command line interface of XDM, 2-9
Commands
    ABL2PLD, 2-20
    ANNOTATE, 2-20
    ASCTOVST, 2-29
    Browse, 2-34, 3-1
    CLEANUP, 2-20
    Cursor, 2-38
    DirClean, 2-34
    Directory, 2-34, C-9, C-10
    DOS, 2-35
    Edit, 2-35, 3-1

entering using keyboard, 2-9
entering using mouse, 2-9
executing with a function key, 2-38
Family, 2-38
FITEQN, 2-24
    example, C-20
FITNET, 2-25, 5-3
JED2PLD, 2-22
KeyCursor, 2-38
Keydef, 2-38
    example, C-9
MAKEJED, 2-31
MAKEPRG, 2-32
    example, C-22
Menucolors, 2-39
Mouse, 2-39
Options, 2-39
PALCONVT, 2-26
Palette, 2-40
Part, 2-40
PINSAVE, 2-18, 2-23, C-20, C-22
PLUSASM, 2-22
Printer, 2-40
PROLINK, 2-33, C-22
Readprofile, 2-40
repeating, 2-9
Saveprofile, 2-40
SDT2NET, 2-21, 5-15
selecting, C-7
Settings, 2-40
Speed, 2-40
using shortcuts, 2-9
Version, 2-37
VMH2VST
    example, C-23
VMH2WIR, 2-32
    example, C-23
VSM, 2-30
    example, C-23
WIR2NET, 2-21, 5-15
XDM, entering, 2-8
XNF2VST, 2-29
XNF2WIR, 2-31
XNFMERGE, 2-23
XSIMMAKE, 2-28
Components
    device-specific, 5-7
    XEPLD library, in schematics, 5-3
Configuring the XEPLD environment, C-9
Conversion of PAL-based designs, 2-26
CUPL, 5-15
    see also PLD compilers
Cursor command, 2-38

D

DATA directory, 5-13
Database file, 5-15
Default options, selecting, 2-39
Defining a function key, 2-38
Deleting extra files, 2-34
Design
    annotation statements, 4-4
    behavioral, C-11
    choosing target device, 2-17, 2-40
    example, C-2
        equation files, C-12
    overview of procedure, 1-1, 5-1
Design Entry menu, 2-18
Design rule checker integrator module, 5-7
Device
    choosing target for design, 2-17, 2-40
    determining speed of, 2-40
    XC7000, C-9, C-10
    XC7200 and XC7300, 2-17
DirClean command, 2-34
Directories, 5-10
    CLIB, 5-13
    DATA, 5-13
    MSG, 5-13
    selecting current directory, 2-34

structure of in XDM, 5-10
TUTORIAL, 5-13, C-9, C-10
viewing structure, 2-34
XACT root directory, 5-13, C-5
Directory command, 2-34, C-9, C-10
Directory settings field, 2-17, C-9, C-10
Display Manager, 2-4
Divider integrator module, 5-9
DOS command
entering from within XDM, 2-35
Downloading HEX files, 2-33
DS120 programmer, C-22

E

Edit command, 2-35, 3-1
Editing files, 2-35
EDITOR environment variable, C-5
Environment variables
EDITOR, C-5
XACT, C-5
Environment, configuring, C-9
EQN file, 3-33
Equation file
assembling, 2-22
examples, C-12
Equations
ADD, 4-44
ADDMODE, 4-46
ALU format, 4-40, 4-41
CLKF, 4-48
combinatorial and registered, 4-40
control, 4-43
defining signal polarity, 4-66
EXPORT, 4-52
FBK, 4-54
FI, 4-56
for example design, C-12
PIN, 4-57
PRLD, 4-58
RSTF, 4-48, 4-59
SETF, 4-48, 4-60
SHIFT, 4-61
TRST, 4-65
ERR file, 5-1, 5-16
Errors, 5-16
Example design, C-2
Example files, C-4
ABEL, C-11
EXE files, 5-13
Executable files, 5-13
Exiting
temporarily, to DOS, 2-35
XDM, 2-3, C-6
EXPORT equation, 4-52

F

Family command, 2-38
Family settings field, 2-17, C-9, C-10
Fast Output Enable see FOE
FastCLK
design rules, 5-8
pin, 4-10
FASTCLOCK statement, 4-10
FastInput (FI)
pins, 4-12, 4-56
source equation, 4-56
FASTINPUT statement, 4-12
FBK equation, 4-54
FFB product term assignment equation (EXPORT), 4-52
FI equation, 4-56
Fields
settings, 2-17
Files, 5-10
ABEL examples, C-11
BAT, 5-13
database, 5-15
deleting extra, 2-34
editing, 2-35
equation, assembling, 2-22
ERR, 5-1, 5-16
EXE, 5-13
HEX, 5-16, C-22

creating, 2-32
downloading, 2-33
JEDEC, 5-15, 5-16
creating, 2-31
LOG, 5-1
log, 5-16
MAP, C-21
netlist, 5-3, 5-15
PAR, C-22
PDS, 5-15
PIN, C-21
PLD, 5-15
PLUSASM equation, 5-15
PRG, 5-16, C-22
PROGLIST.XDM, 2-1
program list, 2-1
programming, 5-16, C-22
RCVR.ABL, C-14
RCVR.PDS, C-12
report, 5-15, C-21
RES, C-21
tutorial example, C-4
TXT, 5-13
UART_EQN.PLD, C-17, C-18
viewing, 2-34, 2-35
VMF, 2-18, 2-23, C-20, C-22
VMH or VMD, 5-15
VST, 2-29, C-23
WIR, C-23
wirelist, 2-30, 2-31, 2-32
XDM.PRO
reading, 2-40
writing, 2-40
XEPLD.CFG, 5-13
FITEQN command, 2-24
example, C-20
FITNET command, 2-25, 5-3
Fitter
invoking, C-20
results (report), 3-2
Fitter menu, 2-24
Flip-flops
optimization, 5-9
FOE
design rules, 5-8
pin option, 4-19, 4-30
pins, 4-13
FOEPIN statement, 4-13
Freezing pins, 2-24, 2-25, C-20, C-22
Function key, defining action of, 2-38

G

Gate optimizer, 5-9
Graphic interface of XDM, 2-9

H

HEX files, 5-16, C-22
creating, 2-32
downloading, 2-33

I

I/O resource report, 3-2
Importing a JEDEC file, 2-22
INCLUDE_EQN statement, 4-15
Input signal polarity, 4-67
INPUTPIN statement, 4-16
Inputs
hanging, 5-7
Integration
freezing pins, 2-24, 2-25
of a behavioral design, 2-24
of a PAL-based design, 2-26
of a schematic netlist, 2-25
Integrator
chip builder module, 5-7
design rule checker, 5-7
divider module, 5-9
interconnector module, 5-9
mapper module, 5-9
minimizer, 5-5
muncher, 5-3
netlist reader module, 5-3
optimizer module, 5-9

Partitioner module, 5-5
using on example design, C-20
Interconnector integrator module, 5-9
Interface
command line, 2-9
graphic, 2-9
Inverters, optimization, 5-9
IOPIN statement, 4-18

J

JED2PLD command, 2-22
JEDEC
importer, 5-15
JEDEC file, 5-15, 5-16
creating, 2-31
importing into design, 2-22

K

KeyCursor command, 2-38
Keydef command, 2-38
example, C-9

L

Library
directory contents, 5-13
Local feedback equation (FBK), 4-54
Local shift equation (SHIFT), 4-61
LOG file, 5-1
Log files, 5-16
Logic Optimizer report, 3-16
Logic resource report, 3-2

M

Main menu, C-6, C-7
MAKEJED command, 2-31
MAKEPRG command, 2-32
example, C-22
MAP mapping report file, C-21
Mapper integrator module, 5-9
Mapping report, 3-4, C-21
Menucolors command, 2-39
Menus, 2-16
Design Entry, 2-18
Fitter, 2-24
main, 2-3, C-6, C-7
opening, 2-9, 2-17
Profile, 2-38
Translate, 2-18
Utilities, 2-33
Verify, 2-28
Merging XNF files, 2-23
Messages, 5-16
directory, 5-13
MINIMIZE statement, 4-23
Minimizer integrator module, 5-5
turning off for an equation, 5-7
Mode/connection port, 2-18
Model for simulation, creating, C-23
Motif, 2-4
mouse
configuration in XDM, 2-5
Mouse button
configuration in XDM, 2-9, C-8
functions, changing, 2-39
settings, 2-9, C-8
Mouse command, 2-39
Mouse settings field, 2-18
MSG directory, 5-13
Muncher integrator module, 3-2, 5-3
mwmrc file, 2-4

N

NAND gates
optimization, 5-9
Navigating through directories, 2-34
Netlist
file, 5-3, 5-15
integrating, 2-25
merging multiple files, 2-23
OrCAD, creating, 2-21
reader integrator module, 5-3
reading, 5-3
Workview, creating, 2-21
NODE statement, 4-25
NOT optimization, 5-9

O

Opening a menu, 2-9, 2-17
Openwindows, 2-4
Optimizer integrator module, 3-2, 5-9
Options command, 2-39
OrCAD
  file format conversion, 2-29
  netlist, creating, 2-21
  VST simulation file, 2-29
Output polarity, 4-66
OUTPUTPIN statement, 4-30
Overlapping objects
  cleaning up, 2-20
Overview, 1-1, 5-1
  behavioral design entry, 1-9, 1-10
  schematic capture, 1-7
  XDM, 1-2

P

PAL Interconnect report, 3-29
PALCONVT command, 2-26
Palette command, 2-40
PAR partitioner report file, C-22
Part
  determining speed of, 2-40
  family, selecting, 2-17, 2-38
Part command, 2-40
Part settings field, 2-17
PARTITION statement, 4-32
  example of, C-19
Partitioner
  report, C-22
Partitioner integrator module, 5-5
Partitioner Log report, 3-22
Partitioner report, 3-9
PATH
  statement for XDM, 2-2, 2-4
PATH, and XACT directory, C-5
PDS files, 5-15
Pin assignment, C-22
  example of, C-19
Pin declaration statements
  CEPIN, 4-6
  FASTCLOCK, 4-10
  FOEPIN, 4-13
  INPUTPIN, 4-16
  IOPIN, 4-18
  OUTPUTPIN, 4-30
PIN equation, 4-57
Pin freezing, 2-24, 2-25, C-20, C-22
PIN pinlist report file, C-21
PINFBK pin option, 4-18
Pinlist
  report, 3-7, C-21
Pins
  FastCLK, 4-10
  FastInput, 4-12, 4-56
  FOE, 4-13
  FOE option, 4-19, 4-30
  I/O, defining, 4-18
  input source equation (PIN), 4-57
  input, defining, 4-16
  names,report, 3-7
  PINFBK option, 4-18
  PINTRST option, 4-30
  usage report, 3-5
PINSAVE command, 2-18, 2-23, C-20, C-22
PINTRST
  pin option, 4-19, 4-30
PLD
  assembling equations for, 2-22
  compilers, 5-15, C-11
  files, 5-15
  pin position labels, 4-8
PLUSASM
  assembler
    partitioner and minimizer, 5-5
  command, 2-22
  declaration section, 4-4
  delimiters, 4-68
  equation files, 5-15

files, equation section, 4-40
operators and special characters, 4-68
reserved characters, 4-69
syntax, 4-72
Polarity
of signals, defining, 4-66
Port components
design rules, 5-8
Power-on reset, 4-58
Preload equation (PRLD), 4-58
PRG files, 5-16, C-22
Printer command, 2-40
PRLD equation, 4-58
Product terms
partitioner and minimizer, 5-4, 5-6
Profile menu, 2-17, 2-38
PROGLIST.XDM, 2-1
proglist.xdm file, 2-6
Program list file, 2-1
Programming files, 5-16, C-22
PROLINK command, 2-33, C-22
PWR statement, 4-38

Q

Quitting XDM, 2-3, C-6

R

RCVR.ABL file, contents, C-14
RCVR.PDS file, contents, C-12
Readprofile command, 2-40
Reference designators, updating, 2-20
Register clock source (CLKF), 4-48
Registers
optimization, 5-9
Removing files, 2-34
Repeating commands, 2-9
Report files, 5-15, C-21
Reports
EQN file, 3-33
Logic Optimizer, 3-16
Mapping, 3-4, C-21
optimized outputs, 3-17
PAL Interconnect, 3-29
Partitioner, 3-9, C-22
Partitioner Log, 3-22
Pinlist, 3-7, C-21
pins and product term allocation, 3-22, 3-23
Resource, 3-2, C-21
viewing, 3-1
RES resource report file, C-21
Resources
munching unused, 5-3
RSTF equation, 4-48, 4-59
Rules for XEPLD designs, 5-7
device-specific components, 5-7
FastCLK, CE, and FOE, 5-8
hanging inputs, 5-7
port components, 5-8

S

Saveprofile command, 2-40
Saving XDM settings, 2-40
Schematics
integrating designs, 2-25
reading netlists, 5-3
schematic capture overview, 1-7
SDT2NET command, 2-21, 5-15
SETF equation, 4-48, 4-60
Settings
mouse button, C-8
saving, 2-40
Settings command, 2-40
Settings fields, 2-17
Shape of cursor, changing, 2-38
Shell for XEPLD software, 2-1
SHIFT equation, 4-61
Shortcuts for command entry, 2-9
Signals
active low, 4-66
defining polarity, 4-66

Signature
    specifying for programming files, 2-31, 2-32
Simulation
    automatic file creation, 2-28
    creating model for, C-23
Size Factor
    report, 3-10
Speed command, 2-40
Starting XDM, 2-2, 2-6, C-6
STRING statement, 4-39
Sun 4, 2-4
Supported software, displaying list of, 2-37

T

Target device, choosing, 2-17, 2-40
Text editor, accessing from XDM, C-10
Translate menu, 2-18
Translator see Integrator
Tri-state
    control equation (TRST), 4-65
TRST equation, 4-65
Tutorial
    example design, C-2
    example files, C-4
TUTORIAL directory, 5-13, C-9, C-10
TXT files, 5-13

U

UART
    example design, C-2
UART_EQN.PLD file, C-17, C-18
UIM
    and interconnector integrator module, 5-9
Unused resources, munching, 5-3
Utilities menu, 2-17, 2-33

V

Variables
    EDITOR, C-5
    XACT, C-5
Verify menu, 2-28
Version command, 2-37
Viewing files, 2-34, 2-35
Viewsim wirelist file, 2-30, 2-31, 2-32
VMD file, 5-15
VMF file, 2-18, 2-23, C-20, C-22
VMH file, 5-15
VMH2VST command
    example, C-23
VMH2WIR command, 2-32
    example, C-23
VSM command, 2-30
    example, C-23
VST file, 2-29, C-23

W

Warnings, 5-16
window accelerators
    in XDM, 2-6
window buttons
    in XDM, 2-5
window operations
    in XDM, 2-5
WIR file, C-23
WIR2NET command, 2-21, 5-15
Wirelist file, 2-30, 2-31, 2-32
Workview
    netlist, creating, 2-21
    see also Viewlogic

X

XACT
    root directory, 5-13, C-5
    software directory, 2-2
    variable, C-5
XACT Design Manager, see XDM
XC7000 devices, C-9, C-10
XC7200 and XC7300 devices, 2-17
xdefaults file, 2-4

XDM, 2-1
  accessing text editor from XDM, C-10
  command entry, 2-8
  command line interface, 2-9
  configuring the environment, C-9
  displaying settings of, 2-40
  exiting, 2-3
  graphic interface, 2-9, C-8
  handling unexpected results, C-8
  main menu, 2-3, C-6, C-7
  menus, 2-16
    Design Entry, 2-18
    Place Route, 2-24
    Profile, 2-17
    Translate, 2-18
    Utilities, 2-17, 2-33
    Verify, 2-28
  mouse configuration, 2-9, C-8
  obtaining help, 2-36
  overview of, 1-2
  PATH statement for, 2-2, 2-4
  proglist.xdm file, 2-6
  quitting, 2-3, C-6
  saving settings, 2-40
  starting, 2-2, 2-6, C-6
  suspending, 2-4
  use with X-windows, 2-1
  user interface, 2-8, C-7
  workstations, 2-4
    active window, 2-6
    configuring X–Windows, 2-4
    mouse configuration, 2-5
    obtaining help, 2-36
    window accelerators, 2-6
    window buttons, 2-5
    window operations, 2-5
  X–Windows, 2-6
XDM.PRO file, 2-2
  displaying contents of, 2-40
  reading, 2-40
  writing, 2-40
XEPLD
  devices, 2-17
  overview, 1-1, 5-1
XEPLD.CFG file, 5-13
XNF files
  merging, 2-23
XNF2VST command, 2-29
XNF2WIR command, 2-31
XNFMERGE command, 2-23
XSIMMAKE command, 2-28
X–terminal windows, 2-5
X–Windows
  in XDM, 2-1, 2-4

Trademark Information