XILINX®

ONLINE

XEPLD SCHEMATIC DESIGN GUIDE

● TABLE OF CONTENTS

● INDEX

GO TO OTHER BOOKS

XACT™ STEP

XEPLD Schematic Design Guide

Getting Started with Schematic Design

Design Entry Techniques

Controlling Design Implementation

Design Applications

Common Questions and Answers

Attributes

Library Selection Guide

Design Implementation and Simulation

Preface

About This Manual

This Schematic Design Guide provides information on using the XEPLD Pre-Release software and supported CAE interfaces to create designs for Xilinx CPLD devices. It focuses on schematic design techniques, including making the best use of library components in schematics. For more detailed information about using XEPLD™ with CAE tool interfaces, refer to the following:

- *Libraries Guide*

For related information about CPLD design entry, refer also to the following:

- *Synthesis Design Guide*
- *EZTag User Guide*

Manual Contents

This manual covers the following topics:

- Chapter 1, “Getting Started with Schematic Design,” presents an overview of schematic design for CPLD devices, including a simple example design.
- Chapter 2, “Device Entry Techniques,” describes the fundamental techniques for expressing logic in a schematic design for a CPLD device.

- Chapter 3, “Controlling Design Implementation,” discusses techniques for controlling how various parts of your design are implemented into a CPLD device.
- Chapter 4, “Design Applications,” describes useful techniques for expressing efficient CPLD designs.
- Appendix A, “Common Questions and Answers,” lists common problems in processing CPLD designs and explains their most likely causes and solutions.
- Appendix B, “Attributes,” lists and describes CPLD schematic attributes, which allow access to CPLD architectural features.
- Appendix C, “Library Selection Guide,” lists all the symbols that may be used in CPLD schematic designs.
- Appendix D, “Design Implementation and Simulation,” describes the xepld used to invoke the fitter, and the xepldsim commands used to prepare functional and timing simulation models.

Conventions

The following conventions are used in this manual's syntactical statements:

`Courier font regular`	System messages or program files appear in regular Courier font.
`Courier font bold`	Literal commands that you must enter in syntax statements are in bold Courier font.
italic font	Variables that you replace in syntax statements are in italic font.
`[ ]`	Square brackets denote optional items or parameters. However, in bus specifications, such as bus [7:0], they are required.
`{ }`	Braces enclose a list of items from which you must choose one or more.
. . .	A vertical ellipsis indicates material that has been omitted.
. . .	A horizontal ellipsis indicates that the preceding can be repeated one or more times.
|	A vertical bar separates items in a list of choices.
↵	This symbol denotes a carriage return.
→	This symbol denotes hierarchy in menu commands.

Table of Contents

Preface

About This Manual iii
Manual Contents iii

Conventions

Chapter 1 Getting Started with Schematic Design

An Overview of Schematic Design Methods 1-1
Schematic Design Flow Example 1-3
  Step 1 — Configure the Design Entry Tool 1-4
    On Workstations 1-4
    On PCs with Viewlogic PRO Series 1-5
    Orcad 386+ Procedure 1-6
  Step 2— Draw the Design 1-7
  Step 3— Perform Functional Simulation (Optional) 1-8
    On Workstations 1-8
    On PCs with Viewlogic PRO Series 1-9
    On PCs with OrCAD 386+ 1-10
  Step 4— Implement the Design 1-11
    On Workstations 1-11
    On PCs 1-12
  Step 5 — Examine the Reports 1-14
  Step 6 — Timing Simulation 1-15
    On Workstations 1-15
    On PCs with Viewlogic PRO Series 1-15
    On PCs with OrCAD 1-16
  Step 7 — Device Programming 1-16

Chapter 2 Design Entry Techniques

Library Symbols 2-1
  Specific Components 2-2
  Primitives, Macros and User-generated Symbols 2-2
  Power and Ground Signals 2-3
Input/Output Buffers 2-4
  Inputs, Outputs, and Bidirectionals 2-4
  Three-State Multiplexing 2-5
  Clock Inputs 2-7
  Output Enable Signals 2-9

Asynchronous Clear and Preset 2-10
Behavioral Modules 2-11
Compiling Behavioral Logic 2-12
Behavioral Module Symbols in Schematics 2-13
Behavioral Module Example for Viewlogic 2-14
Behavioral Modules and Three-State Outputs 2-15
Using SymGen to Create Custom Symbols from Behavior Modules 2-17
Viewlogic Symbols 2-17
OrCAD Symbols 2-17
Storing Custom Components 2-18
Viewlogic Components 2-18
OrCAD Components 2-19
Hierarchical Design 2-19
Custom Macro Example for Viewlogic 2-20
Retargeting a Design From a Different Family 2-20
XC7000 to XC9000 Design Migration Issues 2-21
Schematic Conversion Procedure 2-22
Using Viewlogic on Workstations 2-23
Using Viewlogic PRO Series on PC 2-24
Using OrCAD on PC 2-24
Processing a Design After Conversion 2-25
Attribute Compatibility 2-25
Converting Behavioral Modules 2-26

Chapter 3 Controlling Design Implementation

Target Device Selection 3-1
For Workstations 3-1
For PCs 3-2
Software Device Selection Criteria 3-3
Controlling Register Initial State 3-4
Controlling Power Consumption 3-4
Changing Power Mode for a Specific Component 3-4
Changing Global Power Mode on Workstation 3-5
Changing Global Power Mode on PCs 3-5
Controlling Output Slew Rate 3-6
Controlling the Pinout 3-6
Pin Freezing 3-7
Freezing Pins on a Workstation 3-7
Freezing Pins on a PC 3-7
Guide Files 3-7

Pin Assignment ... 3-8
Ignoring the LOC Attribute on a Workstation ... 3-9
Ignoring the LOC Attribute on a PC ... 3-9
Pin Assignment Precautions ... 3-9
Controlling Logic Optimization ... 3-10
Collapsing Product Term Limit ... 3-10
Controlling Pterm Limits on a Workstation ... 3-11
Controlling Pterm Limits on a PC ... 3-11
If the Path Delay is Not Satisfactory ... 3-11
Preventing Collapsing of a Logic Node ... 3-12
Controlling Timing Paths ... 3-13
Timing Optimization ... 3-13
XACT Performance ... 3-14
Timing Definitions ... 3-14
The TIMESPEC Symbol ... 3-16
Defining Timing Path End Points ... 3-17
Using Predefined Groups ... 3-17
Specifying Time Delay Units ... 3-18
Disabling Timing Specifications ... 3-18
Reducing Levels of Logic ... 3-19

Chapter 4 Design Applications

Read-Back Registers ... 4-1
Bidirectional Signals and Buses ... 4-2
Multiplexing 3-State Signals ... 4-2
Combinational Feedback Loops ... 4-4

Appendix A Common Questions and Answers

Drawing the Design ... A-1
Why Do I See White Boxes Instead of Symbols? ... A-1
Fitting the Design ... A-2
What Does “Unrecognized Symbol” Mean? ... A-2
Simulating the Design ... A-2
Why Are My Registers Stuck at the Preload Value? ... A-2
Why Are My Internal Nodes Not Visible During Timing Simulation? ... A-3

Appendix B Attributes

Target Device Selection Attribute — PART ... B-1
Viewlogic Procedure ... B-2
Behavioral Module Attributes — FILE and DEF ... B-2

Pin Assignment Attribute — LOC B-2
Power Setting Attribute — LOWPWR B-3
Logic Optimization Attribute — OPT=OFF B-3
Register Preload State — INIT B-4
Output Slew Rate — FAST B-4
Minimization of Redundant Logic — MINIM B-4
Timing Specification Attributes — TSnn and TNM B-4

Appendix C XEPLD Library Selection Guide

Appendix D Design Implementation and Simulation

Running the Fitter D-1
From a Workstation D-1
XEPLD Command Parameters D-2
From a PC D-3
Generated Reports D-4
Functional and Timing Simulation D-5
Simulation on a Workstation D-5
Simulation on a PC Under Windows D-6
Simulation on a PC using DOS D-7
Simulating Power-On Initialization D-7

Chapter 1

Getting Started with Schematic Design

This chapter will help you quickly understand how to develop a schematic design using the CPLD design implementation software (fitter). A brief schematic design example is included, illustrating device-independent schematic design entry and simulation processes.

An Overview of Schematic Design Methods

A schematic design defines the functionality of a logic circuit using one or more schematic files, each of which contains components from a Xilinx-supplied library, such as gates, flip-flops and building-block functions similar to 74xx TTL devices. Schematics can also contain “custom” symbols for which you define the functionality using behavioral modules (similar to PAL devices). Behavioral modules are discussed fully in Chapter 2. Figure 1-1 summarizes the design flow.

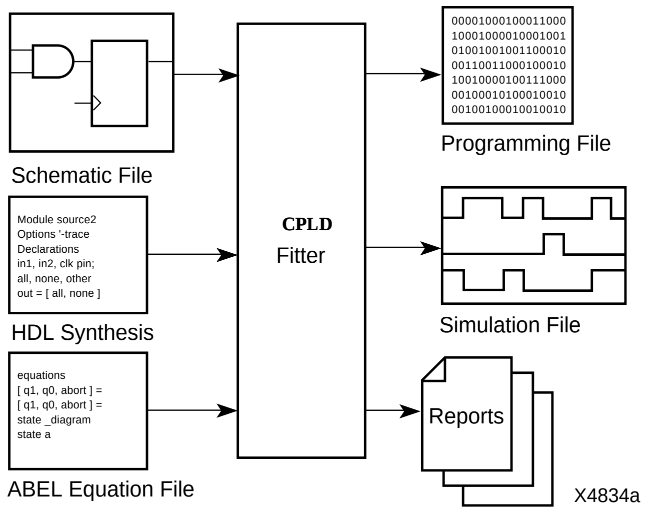


Figure 1-1 Basic Schematic Design Flow

Currently, the Viewlogic and OrCAD 386+ software packages are directly supported by Xilinx CPLD library and interface for CPLD design entry and simulation. Other compatible interfaces and CPLD libraries may be available from their manufacturers.

Schematic Design Flow Example

This section runs through the entire schematic design process, from creating a design to programming and simulating the design. The following device-independent design, a 4-bit Johnson counter, is used as an example:

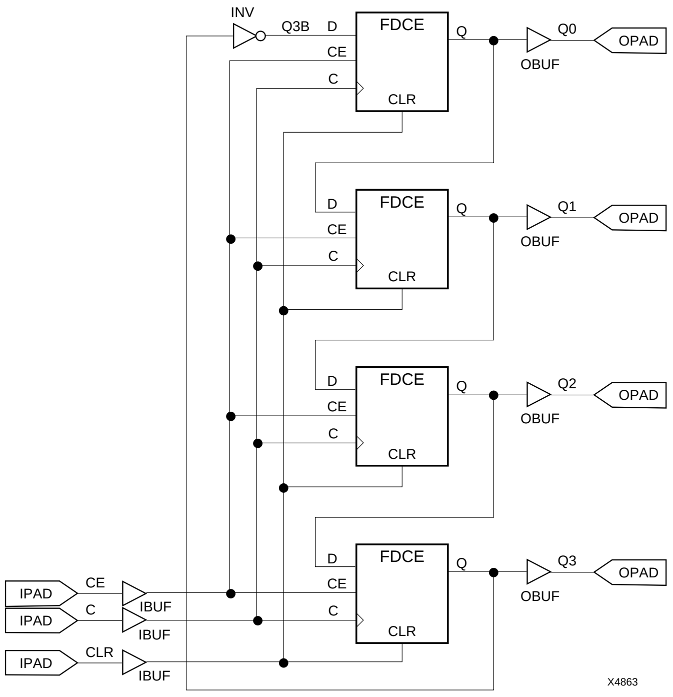


Figure 1-2 Example 4-Bit Johnson Counter Design

Simulation results for this design are shown in Figure 1-3.

The design entry and simulation steps are summarized for Viewlogic and OrCAD software. Other supported schematic design software will have similar procedures; refer to the manufacturer's documentation.

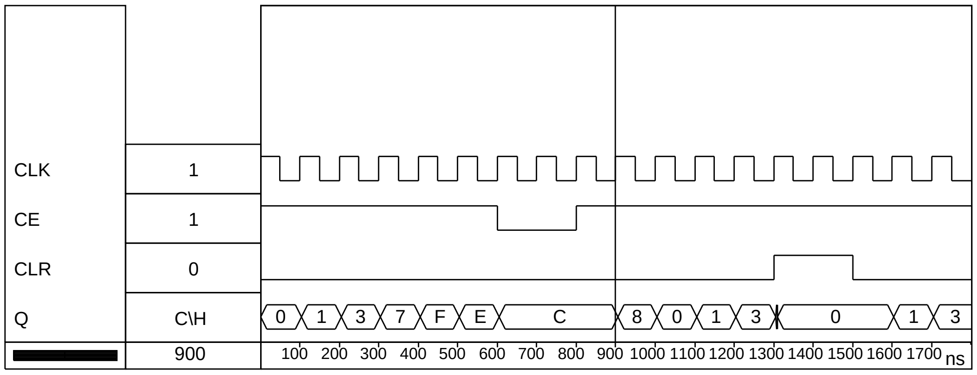


Figure 1-3 Example Viewlogic Functional Simulation Results

Step 1 — Configure the Design Entry Tool

On Workstations

Many design entry tools have a project management facility that you can use to create a working directory for your design. In Viewlogic's Powerview, you would use the **`Project -> Create`** command.

Next, you will need to configure the design entry tool to access the CPLD component library for schematics you develop in the project you just created. In Powerview, you would use the **`Project -> Search Order`** command to open a dialog window listing the configured libraries. On the first available library line, enter the directory path where the CPLD Viewlogic library is installed on your system. For example, enter *installation_path*/**unified/xc9000** (for XC9000 target devices) or *installation_path*/**unified/xc7000** (for XC7000 target devices), where *installation_path* is the root directory where the CPLD software package was installed. Under the "Library" column, enter **XC9000** or **XC7000**, which is also known as the library alias. Under the Type column, select **Read** (read-only format).

If you are not using the Viewlogic project manager, you can make a copy of the viewdraw.ini file in your project directory (copied from the Viewlogic standard directory) and add one of the following lines to the end of the file:

`DIR [r]` *installation_path***`/unified/xc9000`** (xc9000)

DIR [r] *installation_path***/unified/xc7000** (xc7000)

where *installation_path* is the root directory where the CPLD fitter package was installed.

If you plan to simulate using Viewsim, you should also include the Viewlogic "builtin" library in the **Search Order** window or the viewdraw.ini file.

On PCs with Viewlogic PRO Series

Create a directory for your design. If you are using PROflow, your Xilinx CPLD libraries will be configured for you automatically whenever you create a new project. Otherwise, use the **Project Manager** and **Library List Editor** utilities found in ProCapture.

If you use the Project Manager, the Xilinx CPLD library required for this example is installed under the path *installation_path*\unified\xc9000 (for XC9000 target devices) or *installation_path*\unified\xc7000 (for XC7000 target devices). It is in megafile format. The required viewdraw.ini alias is either xc9000 or xc7000 depending on the CPLD family you are targeting. If you plan to simulate, you should also include the Viewlogic "builtin" library.

Enter PROflow by selecting the **PROflow** icon. From PROflow:

Select **Design Entry**, then select the **Project Manager** button**.**

To create a directory for your project, press the **Create** button and select a directory for your project.

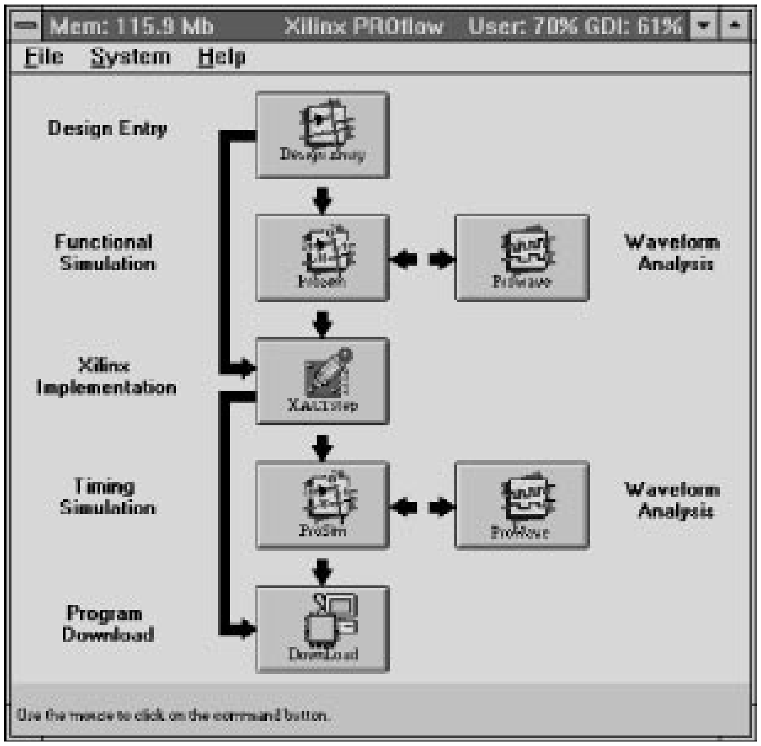


Figure 1-4 Xilinx PROflow

From **`Design Entry`**, click on **`Select Family`**.

Select **`XC9000`** or **`XC7000`**, then select **`OK`**.

Under **`Design Type`**, select **`Schematic`**.

In the **`Design Name`** line, type **`JCOUNT.1`**

Select **`OK`**. PROcapture will be invoked and a design entry window will appear.

Orcad 386+ Procedure

Enter OrCAD. The OrCAD 386+ software described here is not Windows compatible. If necessary, exit Windows, then type at the DOS prompt:

```
orcad
```

Note: Double-clicking on an **`MS-DOS`** icon does not exit you from Windows; it merely opens a DOS window. You must exit Windows completely using **`File`** → **`Exit Windows`**.

Double click on **`Design Management Tools`**. Click on **`Create Design`**.

In the prompt box that appears, type **`jcount`**↵, then click on **`OK`**.

The JCOUNT design appears, highlighted, in the list. Click on **`Suspend to System`**. To target an XC7000 device, type the following at the DOS prompt:

```
xdraft 7
```

To target and XC9000 device, type

```
xdraft 9
```

(wait for the command to complete)

```
exit
```

You are back in the Design Management Tools window. Click on **`OK`** to exit.

Step 2— Draw the Design

Invoke the schematic drawing tool and draw the design. If you are using Powerview, you would invoke the ViewDraw tool. If you are using Viewlogic PRO Series, you would invoke **`PROcapture`**. If you are using OrCAD 386+, invoke **`Schematic Design Tools`**, then **`Draft`**.

If you prefer to use the completed schematic shown in Figure 1-2 as a sample design, you can copy the jcount schematic file from the software directory. For Viewlogic, copy all files and subdirectories under the `*installation_path*\tutorial\vwlogic\jcount` directory into your local directory (the schematic file is `jcount.1` under the `sch` subdirectory). For OrCAD, copy all files under the `*installation_path*\tutorial\orcad\jcount` directory into your local directory.

When drawing a schematic representing a CPLD device, or any sub-sheet in a CPLD design, you should not use any symbols from any other library than the Xilinx CPLD library. Be careful not to use symbols from the Viewlogic builtin library. You may, however, create your own custom symbols representing sub-sheets (hierarchical schematics) or behavioral modules, as described in Chapter 2.

It is important that you label the nets that represent device input/output pins in your design. These are the nets connecting between IPAD and IBUF symbols, and between OBUF and OPAD symbols. These names will appear in the fitter reports and will be used as signal names during simulation.

Save your schematic when finished. The Viewdraw **Write** command (**`File`** → **`Save As`** in PROcapture) performs schematic rule checking and writes a "wire-list" file in the wir directory (wir/jcount.1). Viewlogic wire-list files and OrCAD schematic files (jcount.sch) are read directly by the CPLD fitter software. For other schematic entry tools, the schematic must be translated into an XNF or EDIF 2.0.0 formatted netlist, which can be read by the XEPLD fitter.

Step 3— Perform Functional Simulation (Optional)

On Workstations

On workstation platforms, functional and timing simulation files are prepared using the xepldsim command provided in the CPLD fitter package. For Viewlogic functional simulation, xepldsim reads your schematic (wire-list file), reads any behavioral module equation files referenced in the design, resolves all attributes affecting functionality (such as INIT), and produces a jcount.vsm file for Viewsim.

To prepare a functional simulation file, type the following in a UNIX command window:

```
xepldsim -vlfunc jcount
```

Your current working directory must be your project directory before executing the xepldsim command.

Invoke Viewsim. Use the network command and enter your design name to read the jcount.vsm file. If you have a simulation command file, you can enter that filename as well on the Viewsim command line. For the JCOUNT sample design, a Viewsim command file can be found in *installation_path*/tutorial/vwlogic/jcount/jcount.cmd. The JCOUNT design is simulated using the following Viewsim commands:

```
vector Q Q[3:0]
wave jcount.wfm CLK CE CLR Q
clock c 1 0
step 50ns
h prld
h CE
l CLR
cycle
l prld
cycle 5
l CE
cycle 2
h CE
cycle 5
h CLR
cycle 2
l CLR
cycle 3
```

The wave command automatically invokes a ViewTrace window which displays the input and output simulation waveforms graphically.

On PCs with Viewlogic PRO Series

Select **`PROsim`** from the **`PROflow`** menu.

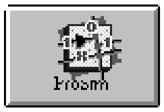


Figure 1-5 ProSim Icon

Click on **`Select Part`**. A menu for selecting parts and packages appears. Select a part and a package from the lists and press **`OK`**.

Place a check in the **`Command File`** box and select **`Browse`**. Go to the `JCOUNT` directory. The file `JCOUNT.CMD` should appear in the files display. Select `JCOUNT.CMD` and press **`OK`**.

Make sure the **`Execute Netlister`** box is checked, then press **`OK.`**

The design is simulated using the following command file:

```
restart
vector Q Q[3:0]
wave jcount.wfm CLK CE CLR Q
clock c 1 0
step 50ns
h prld
h CE
l CLR
cycle
l prld
cycle 5
l CE
cycle 2
h CE
cycle 5
h CLR
cycle 2
l CLR
cycle 3
```

To view the waveform file, go to PROflow and select **PROwave.**

On PCs with OrCAD 386+

To perform functional simulation using the stimulus and trace files provided in the installation directory, follow these steps:

1. From the OrCAD main menu, double-click **Design Management Tools**.
2. Select the **jcount** design.
3. Select **Suspend to System** and type at the DOS prompt:

   xsimmake -f oef6 jcount
4. When **xsimmake** completes, type **exit** to return to OrCAD.
5. Click **OK** to return to main menu.
6. Double Click on **Digital Simulation Tools**.
7. Click once on **Simulate**. Select **Local Configuration** and then **Configure Simulate**.

8. In the Connectivity Database box, change **`jcount.inf`** to **`jcount.vst`**, then select **`OK`**.
9. Double click on **`Simulate`**. A blank simulation waveform window appears with the Q3, Q2, Q1, Q0, CE, CLR and CLK nets listed.
10. Select **`Trace`** → **`Change View`** and enter **`180`**. Select **`Run Simulation`** and enter **`18000`**. The simulation waveforms appear.
11. Exit Simulate using **`Quit`** → **`Abandon Simulation`**.
12. To exit OrCAD, double click on **`To Main`**, then on **`Exit ESP`**. You are now back in DOS. To proceed with design implementation, start Windows, open the **`Xilinx Tools`** icon, and start the **`Design Manager`**.

Step 4— Implement the Design

On Workstations

On workstation platforms, CPLD design implementation is performed using the **`xepld`** command. For Viewlogic designs, **`xepld`** reads the schematic (wire-list file), translates it to an XNF netlist, maps the design to an CPLD device, and produces a programming bit-map (JEDEC) file, fitter report and static timing report.

To implement the jcount design using an XC9000 device, type the following on the UNIX command line:

```
xepld -p 9 jcount
```

To implement jcount using an XC7000 device, type the following:

```
xepld -p 7 jcount
```

Your current working directory must be your project directory before executing the xepld command. The CPLD fitter looks in the current directory for the named Viewlogic schematic (sch/jcount.1 or wir/jcount.1). If a Viewlogic schematic is not found, it looks for an XNF or EDIF netlist file. If the design contains behavioral modules, the fitter reads the associated equation files.

By default, the fitter automatically selects the smallest device in the selected family to meet the needs and constraints of your design. In general, it will select the smallest part and work its way up until it has the part that will satisfy the design.

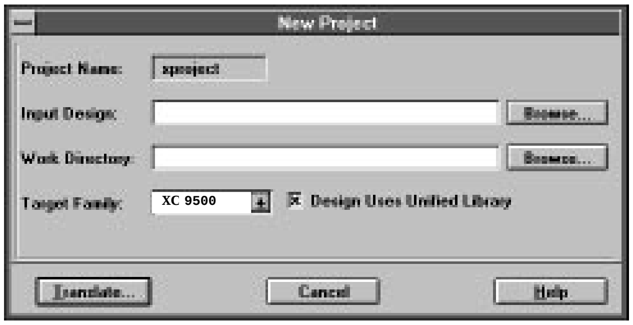
There are several **`xepld`** command-line parameters you can use to control the design implementation process. These are described throughout this Design Guide and summarized in Appendix A.

On PCs

The Windows-based Design Manager is used to fit the design and create a programming file. If you are using Viewlogic PROflow, select **`XACTstep`** from PROflow. Otherwise, select the Design Manager icon from the Xilinx program group. Next, create a new project.

`File → New Project`

The **`New Project`** dialog box will appear. On the **`Input Design`** line, click on **`Browse`**. For Viewlogic schematics, select the `jcount.1` schematic file. For OrCAD schematics, select **`List Files of Type`** → **`OrCAD`**; then select the `jcount.sch` file. Set the **`Target Family`** to **`XC9500`** or **`XC7000`**. On the Work Directory line, click on Browse. Select the project directory you created in Step 1. Then click on **`Translate`**.

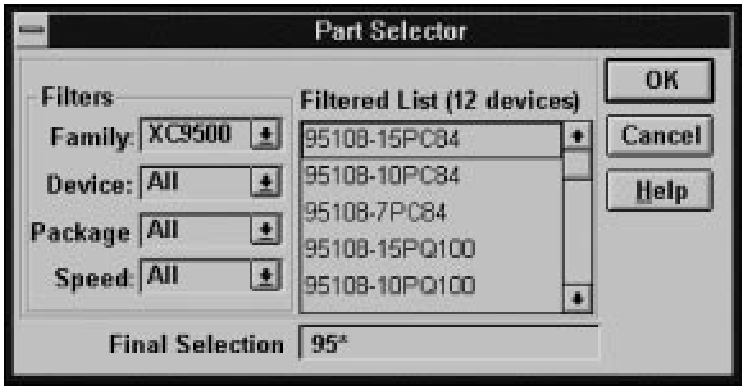


Project Name: xproject
Input Design:
Work Directory:
Target Family: XC 9500
☒ Design Uses Unified Library

Figure 1-6 New Project Window

Remove the checkmark from **`Read Part from Design`**. Click on **`Select Part`**. The Part Selector Dialog Box appears. Under **`Family`**, select **`XC9500`** or **`XC7000`**. Select **`ALL`** for **`Device, Speed`**, and **`Package.`** Then click **`OK`**. Click **`OK`** again to begin translation.

The program will automatically select an economical solution to meet the needs and constraints of your design. In general, it will select the smallest part and work its way up until it has the part that will satisfy the design.



Family: XC9500
Device: All
Package: All
Speed: All

95108-15PC84
95108-10PC84
95108-7PC84
95108-15PQ100
95108-10PQ100

Final Selection: 95*

Figure 1-7 Part Selector Dialog Box

Select **`Tools`** → **`Flow Engine`**. Set **`Stop After`** to **`Bitstream`** and select **`RUN`**. The program optimizes the design, fits the design, creates timing file and programming files all in one step. The interface keeps you updated on the progress of processing.

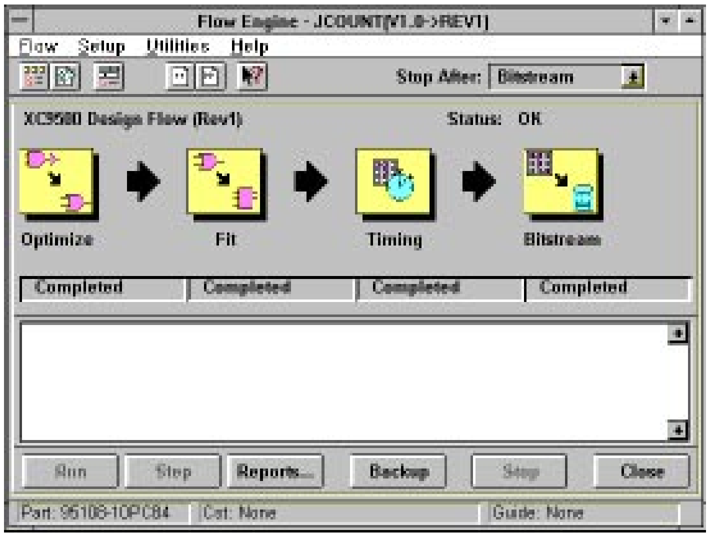


Figure 1-8 Flow Engine

Step 5 — Examine the Reports

Examine the reports to verify that the design was implemented as you expected. The following report files (plain text) are automatically produced by the xepld command. If you are using a workstation, display the contents of the report files listed below. If you are using a PC you may select a report from the report browser as follows:

`Utilities → Report Browser`

or select the report browser icon. The following reports are most useful for schematic designs:

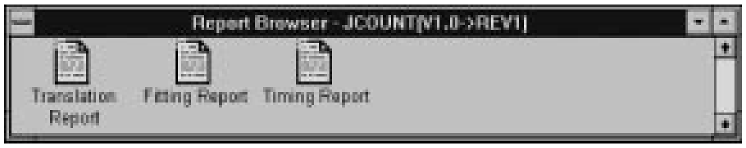


Figure 1-9 Report Browser

- Fitter Report (jcount.rpt)— The fitter report shows the device resources used by the design, how the external nets in your design were mapped to the device pins, and the physical allocation of all device resources.
- Timing Report (jcount.tim) — A timing summary report shows the calculated worst-case timing for the logic paths in your design.

Step 6 — Timing Simulation

On Workstations

The **`xepld`** command automatically creates a timing simulation netlist, jcount_tim.xnf, whenever the design is successfully implemented. For Viewlogic timing simulation, the **`xepldsim`** command reads the jcount_tim.xnf file and produces a jcount.vsm file for Viewsim.

To prepare a timing simulation file, type the following in a UNIX command window:

```
xepldsim -vltime jcount
```

Your current working directory must be your project directory before executing the **`xepldsim`** command.

Invoke Viewsim. Use the **`network`** command and enter the jcount design to read the jcount.vsm file. If you have a simulation command file, you can enter that filename as well. For the JCOUNT sample design, a Viewsim command file can be found in *installation_path*/ tutorial/vwlogic/jcount/jcount.cmd.

The **`xepldsim`** command can also produce timing simulation files for Verilog and for the Synopsys VSS simulators. The **`xepldsim`** command is summarized in Appendix A.

On PCs with Viewlogic PRO Series

From PROflow, select the **`PROsim`** icon from the Timing Simulation block. Put a check in the **`Command File`** box and select **`Browse`**.

Select the `JCOUNT.CMD` file and press **`OK`**. Make sure **`Execute Netlister`** is selected, then press **`OK`**. The simulation runs and generates a waveform which you can view by selecting **`PROwave`**.

The jcount design example is now complete. Select **`File`→ `Exit`** to terminate the PROflow session.

Note: Xilinx has a separate Timing Analyzer for further analysis of timing. See the "EPLD Architecture and Design Trade-offs" chapter for information on the Timing Analyzer, or go to the *XEPLD Reference Guide*.

On PCs with OrCAD

To run a timing simulation on the design:

1. Exit Windows; at the DOS prompt, type:

   ```
   xsimmake -f oet6 jcount
   ```

2. Start **`OrCAD`**, then double click on **`Design Management Tools`**, select **`JCOUNT`** from the list, and click on **`OK`**.
3. Double Click on **`Digital Simulation Tools`**. Double click on **`Simulate`**. A blank simulation waveform window appears with the Q3, Q2, Q1, Q0, CE, CLR and CLK nets listed.
4. Select **`Trace`** → **`Change View`** and enter **`125`**. Select **`Run Simulation`** and enter **`15000`**. The simulation waveforms appear.
5. Exit Simulate using **`Quit`** → **`Abandon Simulation`**.

Step 7 — Device Programming

The fitter automatically creates a JEDEC programming file, jcount.jed, whenever a design is successfully implemented. Once you are satisfied with the results of the fitter (reports and timing simulation), you can download the programming file to the device using the techniques described in the *EZTAG User Guid*e.

Chapter 2

Design Entry Techniques

This chapter discusses the fundamental techniques for expressing logic in a schematic design for CPLDs. It concentrates mainly on the symbols you place in your schematic and how you interconnect them. It also explains how to retarget an existing schematic for an FPGA design to a CPLD, or between the XC7000 and XC9000 families.

Library Symbols

The Xilinx library contains all component symbols used by Xilinx XC7000 and XC9000 device families. While most symbols of the library are common to all families, there are some symbols which are specific to one or more CPLDs.

Physically, each device family has its own schematic library, implemented for each of the supported schematic entry tools. For each tool, there are separate library directories for XC9000 and XC7000 device families. When a library component is supported by multiple device families, its symbol appears in each of the corresponding library directories.

When a component of the same name appears in multiple family libraries, it has the same functionality and graphic symbol body, and similarly named pins. However, the component's implementation, including whether the symbol is a primitive or macro (with underlying schematic), may vary between families. Maintaining consistent functionality and "footprint" facilitates retargeting existing schematic designs between Xilinx device families. The *Libraries Guide* shows the applicability of each library symbol to each of the Xilinx device families.

When drawing a schematic representing an CPLD device, or any sub-sheet in a CPLD design, you should not use any symbols from any other library than the one for your target device family. For example, be careful not to use symbols that belong to a CAE tool's simulation modelling library.

Specific Components

To make your design device-independent, use only the symbols common to all Xilinx device families in which you are interested. The design implementation software automatically maps the symbols in your design onto the chosen target device. Creating a device-independent design allows you to easily test your design with different Xilinx devices.

There are very few library symbols which are specific to only one of the two CPLD families. For example, the BUFGSR symbol is used to explicitly assign device inputs to the global set/reset (GSR) pin of XC9000 devices. These and other special I/O functions are described later in this chapter. Most symbols in each CPLD family library appear in the other library, and most of those are also common to the FPGA families. If you want to create a schematic which can be migrated to different Xilinx device families without modification, you should use the generic IBUF symbol instead of device-specific input buffers (like BUFGSR) and allow the fitter to automatically allocate global set/reset resources.

Primitives, Macros and User-generated Symbols

The following types of symbols can appear in your schematic:

- Library primitives
- Library macros
- User macros (including hierarchical sheet symbols)
- Behavioral modules

The library contains the first two types of components: primitives and library macros. Primitives are those symbols recognized directly by the implementation software such as pads, gates, flip-flops and buffers. Library macros are symbols functionally defined by macro schematics contained in the library. Macro schematics contain primitives and sometimes other macros. Library macros have predefined functionality, but often their implementation is subject to the optimizations performed by the fitter software. Macros are always flattened during schematic-to-netlist translation before the netlist is read by the fitter. The fitter does distinguish primitives contained in macros from those primitives placed in the design schematic.

User macros are custom symbols generated by the user which are functionally defined by user-generated macro schematics. User macro symbols and schematics can be stored in your project directory or in a user library directory that you create. User macro schematics can consist of any mixture of the four types of symbols listed above.

You can create user macros to represent frequently used logic functions and instantiate them in one or more designs. It is often convenient to copy a Xilinx COLD library macro symbol and schematic to your project directory as a template, then rename the symbol and schematic, and modify the symbol pins and schematic to suit your needs. You should not, however, modify any of the Xilinx CPLD library macros themselves or store new user macros in the Xilinx CPLD library directory.

You can add hierarchical structure to a large design by partitioning your logic into multiple schematic sheets. You then create user symbols to represent the schematic sub-sheets in the same manner as you would create user macro symbols. Your hierarchical design is flattened during the schematic-to-netlist translation. Therefore, your schematic partitioning has no effect on the implementation or optimization of your design by the fitter.

Behavioral modules are user-generated custom symbols functionally defined by some logic description other than a schematic. Typically, logic descriptions defining behavioral modules are expressed in Boolean equations or HDL, and processed by a PLD compiler (like XABEL) or a synthesis tool prior to running the fitter. Behavioral modules are discussed later in this chapter.

Power and Ground Signals

Unused inputs on symbols should not be left unconnected. You should never assume a default value for any unconnected symbol input except basic logic gates such as AND or OR. Unconnected AND-gate and OR-gate inputs are simply discarded, as if the gate had fewer inputs.

If a control input to a library macro is left unconnected, the resulting behavior may be different than what you would expect. In some cases, the resulting behavior may be different than if the input were tied either High or Low. If you leave a macro input unconnected, the fitter is usually able to detect it and issue a warning. Both functional and timing simulation will exhibit the actual resulting behavior.

Unused inputs should be tied to a constant High or Low logic level in the schematic. Use the VCC or GND symbol from the library to source a constant logic High or Low value. In most tools, you can also specify a constant High or Low value by connecting a dangling net to the component input pin and then labeling the net as “VCC” or “GND”, which is recognized by the Xilinx software.

Input/Output Buffers

This section discusses techniques for specifying device I/O pins using both general-purpose and special-purpose input/output buffer symbols.

Inputs, Outputs, and Bidirectionals

To represent an ordinary device input, use an IPAD connected to one IBUF symbol. The IBUF can then connect to any number of on-chip logic symbols. An I/O pin of the CPLD device will be allocated to receive the input, and its output driver will be permanently disabled.

To represent an ordinary device output, use an OBUF that is driven by exactly one on-chip logic source. Connect the output of the OBUF to an OPAD symbol. To specify a three-state device output, use an OBUFE or OBUFT instead of the OBUF, and connect its enable/disable input to any on-chip logic source. An I/O pin will be allocated to drive the signal, either always active (if OBUF) or controlled by your enable/disable signal, and the input received by the device pin will be ignored.

To represent a bidirectional device I/O, use an OBUFE or OBUFT whose output is connected to both an IOPAD symbol and to the input of an IBUF, as shown in Figure 2-1.

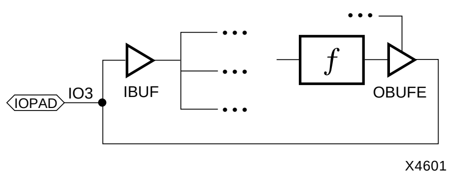


Figure 2-1 Bidirectional I/O Pin

It is important that you label the nets that represent device input/output pins in your design. These are the nets connecting between IPAD and IBUF symbols, and between OBUF and OPAD symbols. These names will appear in the fitter reports and will be used as signal names during simulation.

Note: Do not use the reserved names “PRLD” or “MRESET” as labels for any nets or component instances in your design.

You should not draw “feed-throughs” or duplicate outputs in your design. A feed-through is an IBUF connected directly to an OBUF with no function being performed. A duplicate output is when two or more OBUFs are driven by the same logic source. If you need to express a feed-through or duplicate output, place a BUF symbol in front of each OBUF.

Three-State Multiplexing

CPLD devices can emulate three-state bussing between on-chip signal sources by gating the macrocell feedback to the FastCONNECT structure. (Macrocell feedback signals are never physically in a high-impedance state.) Multiple feedbacks emulating 3-state signals can be wire-ANDed in the FastCONNECT to emulate bussing and 3-state multiplexing. When an on-chip 3-state signal is “disabled”, the macrocell feedback is forced High so that it does not affect the wire-AND. The signal on the wire-AND will therefore follow the logic value of the “enabled” feedback.

To represent 3-state multiplexing (bussing) in the schematic, tie together the outputs of multiple 3-state buffer symbols, like BUFE and BUFT, as in Figure 2-2. You cannot, however, connect such tied signals directly to an output buffer; instead you must pass a tied signal through a logic symbol, like BUF, before driving an output port.

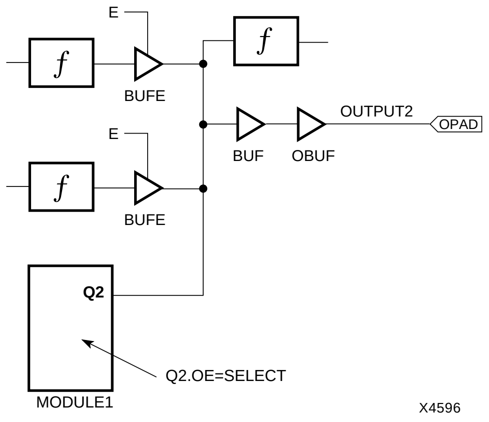


Figure 2-2 Correct On-Chip Three-State Multiplexing

If your design calls for 3-state bussing of multiple signals driven off-chip, it may be better to output each signal source on its own 3-state output pin and tie the pins together off-chip, as shown in Figure 2-3. You cannot connect more than one signal source to the same OBUF or OPAD, as shown in Figure 2-4.

Note: On-chip three-state bussing is also supported by some of the FPGA device families.

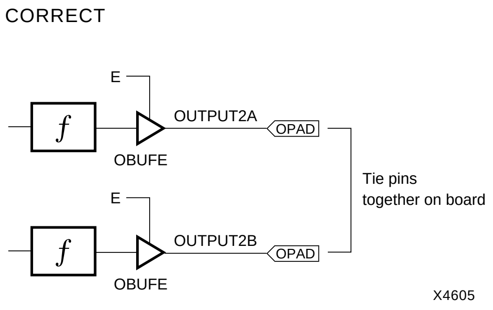


Figure 2-3 Correct Off-Chip Three-state Multiplexing of EPLD Outputs

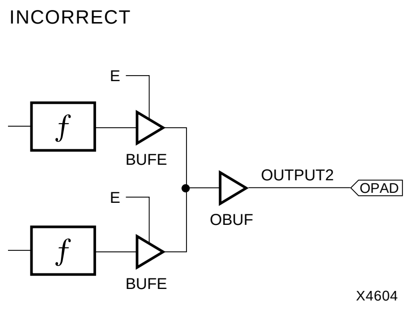


Figure 2-4 Incorrect Three-state Multiplexing of EPLD Outputs

Clock Inputs

To use a device input as a clock source, you can simply connect an IBUF to the clock input of one or more flip-flops in your design. The fitter automatically uses one of the global clock pins (GCK or FCLK) of the CPLD whenever possible.

In XC9000 devices, a global clock input signal may pass through an inverter to perform negative-edge clocking. The same clock input may even be used both inverted and non-inverted to clock different flip-flops on opposite edges of the clock signal, as shown in Figure 2-5. Global clock inputs may also be used as ordinary input signals to other logic elsewhere in the design.

In XC7000 devices, global clock (FastCLK) signals can only be active-High and may not be used for any other logic function in the design.

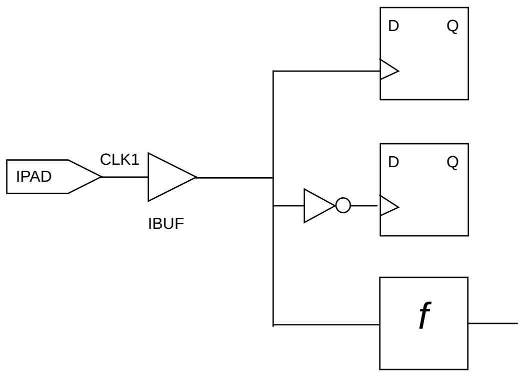


Figure 2-5 Input CLK1 can be Optimized onto a Global Clock Pin (GCK)

If a device input passes through any logic function (other than an inverter) before it is used as a clock, the input cannot be routed directly to the flip-flops using the global clock path. Instead, the flip-flop's clock signal will normally be routed through the logic array.

There are a limited number of global clock pins on each CPLD device (consult the device data sheet). If you need to explicitly control the use of global clock pins, you can use the BUFG symbol in place of an IBUF.

The global clock pins provide much shorter clock-to-output delays than clocks routed through the logic array. Routing a clock through the logic array also uses up one extra p-term for each flip-flop.

You can prevent the fitter from automatically using the global clock pins. On workstations, specify the "-nogck" parameter on the xepld command line as follows:

`xepld -nogck` *design_name*

On PCs, go to the Flow Engine and select:

`Setup → Options`

The Design Implementation Option menu appears. Select:

`Edit Template`

Then select:

`Optimization`

Lastly, place a check on the **`Off`** box adjacent to **`Global Clock`**.

If global clock optimization is disabled, IBUF inputs used as clocks will always pass through the logic array. You can still use BUFG symbols to explicitly specify global clock inputs.

Output Enable Signals

To use a device input to control three-state device outputs, you can simply connect an IBUF to the enable / disable input of one or more OBUFE symbols in your design. The fitter automatically uses one of the global three-state control pins (GTS or FOE) of the CPLD whenever possible.

In XC9000 devices, a global 3-state control input signal may pass through an inverter or control an OBUFT symbol to perform an active-low output-enable. The same 3-state control input may even be used both inverted and non-inverted to enable alternate groups of device outputs, as shown in Figure 2-6. Global 3-state control inputs may also be used as ordinary input signals to other logic elsewhere in the design.

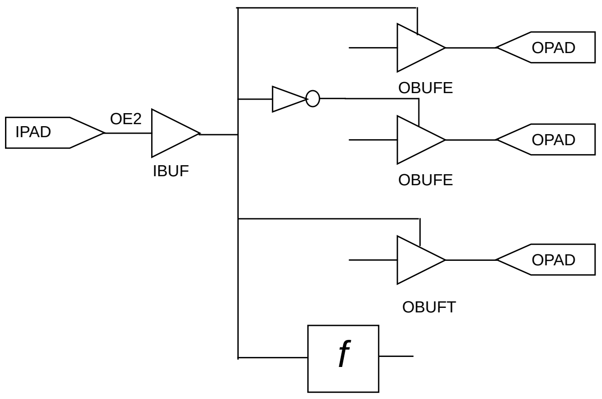


Figure 2-6 Input OE2 can be Optimized onto a Global 3-state Control Pin (GTS)

In XC7000 devices, global output enable (FOE) signals can only be active-High and may not be used for any other logic function in the design.

If a device input passes through any logic function (other than an inverter) before it is used as a 3-state control, the input cannot be routed directly to the device output drivers using the global 3-state control path. Instead, the output enable signal will normally be routed through the logic array.

There are a limited number of global 3-state control pins on each CPLD device (consult the device data sheet). If you need to explicitly control the use of global 3-state control pins, you can use the BUFGTS symbol (for XC9000) or BUFFOE symbol (for XC7000) in place of an IBUF.

The global 3-state control pins provide much shorter input-to-output-enable delays than 3-state controls routed through the logic array. Routing a 3-state control signal through the logic array also uses up one extra p-term for each output.

You can prevent the fitter from automatically using the global 3-state control pins. On workstations, specify the "-nogts" parameter on the xepld command line as follows:

`xepld -nogts` *design_name*

On PCs, go to the Flow Engine and select:

Setup → Options

The Design Implementation Option menu appears. Select:

`Edit Template`

Then select:

`Optimization`

Lastly, place a check on the **`Off`** box adjacent to **`Global Output Enable`**.

If global output enable optimization is disabled, IBUF inputs used for 3-state control will always pass through the logic array. You can still use BUFGTS or BUFFOE symbols to explicitly specify global 3-state control inputs.

Asynchronous Clear and Preset

To use a device input as an asynchronous clear or preset source, you can simply connect an IBUF to the CLR or PRE input of one or more flip-flops in your design. For XC9000 devices, the fitter automatically uses the global set / reset pin (GSR) of the CPLD whenever possible. A global set / reset input signal may pass through an inverter to perform active-low clear or preset. A global set / reset inputs may also be used as an ordinary input signal to other logic elsewhere in the design. (Global set / reset is not available on XC7000 devices.)

If a device input passes through any logic function other than an inverter before it is used as an asynchronous clear or preset, the input cannot be routed to the flip-flop using the global set / reset path. Instead, the flip-flop's clear or preset signal will be routed through the logic array. Routing a clear or preset through the logic array uses up one extra p-term for each flip-flop.

There is only one global set / reset pin on each XC9000 device. If you need to explicitly control the use of the global set / reset pin, you can use the BUFGSR symbol in place of an IBUF.

Note: If a flip-flop has both a clear and preset input and you assert both the clear and preset concurrently, its Q-output is unpredictable. This is because the fitter may arbitrarily invert the logic stored in a flip-flop to achieve better logic optimization. Individual clear and preset operations still produce the correct final logic state as dictated by the user design. Functional simulation does not accurately predict the ultimate behavior of the chip when clear and preset are asserted concurrently. Timing simulation, however, is performed after logic optimization and behaves exactly as the chip will when programmed.

Behavioral Modules

Behavioral modules are user-generated symbols functionally defined by some logic description other than a schematic, typically Boolean equations or HDL. Some reasons why you may want to use behavioral modules in your schematic are:

- If portions of your design are already implemented using conventional programmable logic devices (PLDs), you can re-use your existing PLD equations without having to redraw the same logic schematically.
- It is often easier to express combinational logic functions and state machines using Boolean equations or HDL than using schematics.

- You may wish to express a particular logic function using sum-of-products equations because they will directly map onto the CPLD architecture in that form.

The Xilinx CPLD fitter also accepts entirely behavioral designs which use no schematics. Similar to behavioral modules for schematic designs, behavioral designs are expressed using Boolean equations or HDL and compiled using a PLD compiler (like XABEL) or a logic synthesis tool. Unlike behavioral modules, behavioral designs contain all the device I/O port information in its behavioral description.

This manual describes only schematic-based designs and the behavioral modules which may be contained in them. The procedures for creating behavioral modules in CPLD schematics is essentially the same as for all other Xilinx device families.

Compiling Behavioral Logic

The equation or HDL files defining behavioral modules must be compiled before they can be used by the fitter. There are a variety of PLD compilers and synthesis tools that support design entry for CPLD devices.

Behavioral compilers which are compatible with the CPLD fitter translate their logic descriptions into XNF-formatted netlists or Plusasm-language equation files. Behavioral descriptions expressed directly in the Plusasm equation language can be read directly by the fitter and require no advance compilation. (Plusasm is no longer recommended as an entry language for new designs; it is intended only as an interchange language.)

If the behavioral compiler tool supports the development of completely behavioral designs for Xilinx CPLDs, make sure you select the appropriate mode of operation or compilation flow for producing logic modules, not stand-alone designs. The netlist or equation file produced by the compiler must not contain device I/O pin information. If any of the terminal nodes (inputs or outputs) of your behavioral module are to be connected to CPLD device pins, you must use IBUF and OBUF symbols in your schematic.

If, for example, you are using the XABEL compiler, make sure the **Stand-Alone Design** check box is not checked in the Xilinx EPLD Options dialog box. Then use the **`Compile → Xilinx EPLD`**

`Netlist` command to translate your ABEL design into a Plusasm-language equation file.

If you are using a synthesis tool to prepare a behavioral module, make sure you target a CPLD technology library.

Your compiled behavioral module file is normally stored in your project directory. You can also copy it to a user library directory if you want to use it for more than one project.

Behavioral Module Symbols in Schematics

Using a behavioral module in a schematic design involves creating a symbol to represent your logic, placing the symbol into your schematic and applying necessary attributes to identify the logic-defining file.

1. Use the symbol editing facility of your schematic entry tool to create a symbol representing your behavioral logic. Generally, the name of your symbol will be the name of the behavioral module, although this is not mandatory.

   Place a pin on your symbol for each terminal node (input or output) in your behavioral design that needs to be connected to other logic or I/O ports in your schematic. “Buried” nodes that connect only between logic functions within the behavioral module do not require pins on your symbol.

   Some tools have commands or utilities that automatically generate symbols based on the terminal nodes defined in your behavioral module.

2. Some schematic entry tools distinguish between two types of symbols: primitive symbols (sometimes called “module”) and hierarchical symbols (sometimes called “composite”) that link to schematics beneath them. When creating a symbol to represent a behavioral module, create it as a primitive symbol.

   When your symbol is complete, store it in your project directory or in a user library directory if you want to use the symbol in more than one project.

3. Instantiate the new symbol one or more times in your design schematic and connected it to other logic and I/O buffers as needed. As with library symbols, unused input pins on your

behavioral module symbol should be tied to VCC or GND.

4. Add an attribute to each instance of a behavioral module symbol in your schematic to identify the compiled behavioral logic file. The format of the attribute is

   FILE=*filename*

   where *filename* is the name of the file produced by the behavioral compiler, without extension. If the compiled file is stored in a different directory (such as a user library), include the complete directory path qualification in the FILE attribute.

5. If your compiled behavioral logic file is a Plusasm-language equation file (with extension .pld), you must also apply the **DEF=PLD** attribute to each instance of the behavioral module symbol in your schematic. If your compiled file is an XNF-formatted netlist file (with extension .xnf), you do not need to add the DEF attribute.

Behavioral Module Example for Viewlogic

This simple example shows you how to develop a behavioral module defined by an ABEL-language equation file and represented by a custom symbol in a Viewlogic schematic.

1. Create the following ABEL file named regxor.abl.

```
MODULE regxor
TITLE ‘Registered XOR gate’
regxor device;
IO pin;
I1 pin;
CLK pin;
Q pin istype ‘reg’;
EQUATIONS
Q := IO $ I1;
Q.C = CLK;
end
```

2. Compile the file to create a Plusasm-language equation file named regxor.pld.

   Compile → Xilinx EPLD Netlist

   Make sure the Stand-Alone Design check box is turned off before compiling.

3. In Viewdraw, open a symbol window and create a new symbol named regxor as shown in Figure 2-8. The symbol has three input pins and one output pin corresponding to the pins defined in the ABEL equation file.

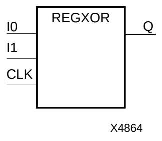


Figure 2-7 The REGXOR Symbol

4. Use the **Change -› Symbol Type -› Module** command to indicate that there is no underlying schematic for the symbol. Then save the symbol in your project directory.
5. Instantiate the symbol in your Viewdraw schematic and connect its input and output pins to logic and/or I/O buffers in your design.
6. Select the regxor component in your schematic and add the attribute **FILE=regxor**. Because the ABEL file was compiled into a Plusasm equation file, also add the **DEF=PLD** attribute to the component.

Behavioral Modules and Three-State Outputs

If you use a behavioral module in your schematic and connect one of its outputs to an output buffer like OBUF, you can enable and disable the device output pin using a three-state control equation in the behavioral module, as shown in Figure 2-8.

Note: This design technique is specific to the XC9000 and XC7000 CPLD families, and may not apply to Xilinx FPGA families.

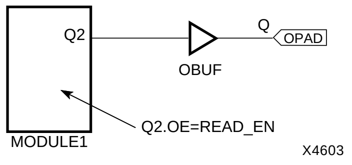


Figure 2-8 Controlling Three-state Output Using a Behavioral Module

If you want to use a three-state behavioral module output to control a bidirectional I/O pin of the device, connect the OBUF output to an IOPAD and IBUF. If the same behavioral module symbol that generates the output is also to receive the I/O pin input, you must use a separate pin of the behavioral module's symbol to receive the signal from the IBUF. Do not tie the signal received from an IBUF to the net driving the OBUF, as shown in Figure 2-9; these input and output nets must remain separate, as shown in Figure 2-10.

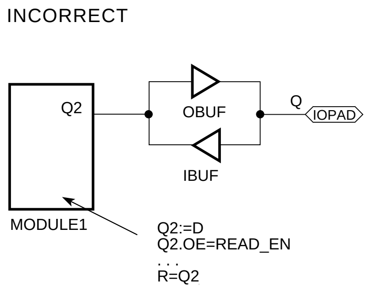


Figure 2-9 Incorrect Way to Connect a Bidirectional Pin

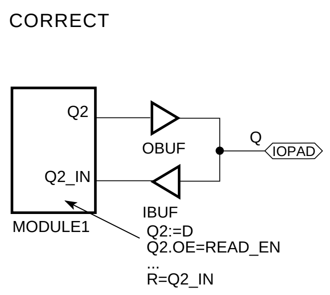


Figure 2-10 Correct Way to Connect a Bidirectional Pin

Using SymGen to Create Custom Symbols from Behavior Modules

On the PC, you can use the Symbol Generation Utility (Symgen) to create a custom symbol for Viewlogic or OrCAD schematics. SymGen reads an .xsf file generated by some supported behavioral compilers, including XABEL.

SymGen is invoked by selecting the **`Symbol Generation Utility`** icon from the Xilinx program group.

SymGen also produces a report, *design_name*.smr, which explains how the symbol was created and displays a diagram of the pinout. The Symbol Generation Utility menu allows you to select either Viewlogic or OrCAD for the type of symbol to generate.

Viewlogic Symbols

If you are a Viewlogic user, SymGen creates the Viewlogic symbol file (*module_name*.1) and places it in the sym directory below your design directory. You can add it to any schematic in your design just as you would any other symbol. No special conversion steps are necessary.

OrCAD Symbols

If you use the Symbol Generation Utility with OrCAD selected as the **`Symbol Type`**, SymGen creates an OrCAD command file (*module_name*.cmd) file and places it in your design directory. To convert the .cmd file into a symbol, you must perform these additional steps:

1. Enter the OrCAD Edit Library utility.

   `File → New → Library`

2. In response to the `Read Library?` prompt, type the following:

   `.\`*library_name*`.lib`

   This creates a library called *library_name* in your design directory.

3. The Library Edit screen appears. Select the **`Import`** command and type *module_name*`.cmd` to invoke the command file.

4. The symbol appears. Select **`Library`** → **`Update Current`** to save the symbol to memory.

5. Select **`Quit`** → **`Update File`** to save the new library file to disk.

6. Select **`Abandon Edits`** to exit the library editor.

To use custom symbols from the new user library file in a schematic, you must add the library name to the schematic tool's configuration. In OrCAD, select **Schematic Design Tools** → **Local Configuration**. Then select and add .*library_name*.lib to the list of libraries. You can now add this symbol to any schematic in your design just as you would any other symbol.

Storing Custom Components

After you create your custom component, you should store it in each design directory where you need to access it.

If you have defined the underlying logic differently for targeting two or more different device families, you should store the component in two or more different project directories or library directories. Each directory would contain the underlying logic for one device family.

Viewlogic Components

You should store your custom library files in your project directory. You cannot add custom symbols to the XC7000 or XC9000 library directory or modify any of the Xilinx-supplied symbols or macros. However, you can copy Xilinx-supplied symbols or macros to your directory as templates, rename them, and edit them.

OrCAD Components

You should normally store a copy of your library file and your macro schematics in each project directory in which you plan to use them. If you prefer to centralize your custom library, you could store your files under the *installation_path*\XC7000 or *installation_path*\XC9000 library directory. Do not add to or modify the xc7000.lib file or any of the library macro schematics supplied by Xilinx.

Hierarchical Design

You can create custom symbols with schematics under them and place these symbols in your top-level or lower-level schematics to create a hierarchical design. This can make your design more modular and easier to understand.

Custom symbols with schematics under them are termed *user macros*, as opposed to *behavioral modules*, which are custom symbols with equations or HDL under them.

The procedure for creating a symbol with an underlying schematic is the same for CPLD and FPGA families:

1. Create a lower-level schematic using CPLD library symbols or other custom symbols. To make a device-independent macro, use only device-independent symbols common to multiple families. If required by your CAE tool, identify the terminal nodes of the macro schematic sheet.
2. Create a symbol for the schematic. The names of pins on your symbol should match the names of the terminal nodes in the underlying schematic.
3. Make sure the symbol is defined as a hierarchical symbol (sometimes called “composite”) as required by your CAE tool.

Custom Macro Example for Viewlogic

This next example shows you how to create a custom macro symbol with an underlying schematic. The steps for Viewlogic users are shown:

1. Create the schematic using common symbols from the CPLD library. For this example, we create a schematic named `regxor`, which should look something like this:

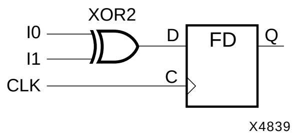


Figure 2-11 The REGXOR Schematic

2. Create a symbol, also named `regxor`, with pin names that match the inputs and outputs of the schematic.

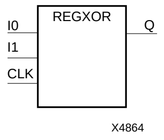


Figure 2-12 The REGXOR Symbol

3. Use the **`Change → Symbol Type → Composite`** command to change the symbol's block type to composite.

Retargeting a Design From a Different Family

When retargeting an existing schematic designed from a different Xilinx device family, there are three aspects of design compatibility which must be considered:

- The symbols in the schematic must be supported by the new target CPLD library.
- Any attributes used in the design must be supported by the CPLD fitter; otherwise, non-applicable attributes should be removed.
- Any behavioral modules used in the design may need to be recompiled to an acceptable file format and using a CPLD

technology library (if applicable).

XC7000 to XC9000 Design Migration Issues

The XC7000 implementation software supports several architecture-specific symbols, attributes and features. This section lists how XC7000-specific features are handled when designs are retargeted to XC9000. In most cases, the CPLD software automatically produces equivalent results.

- In the XC7000 library, IFD symbols are used to target input-pad registers. In XC9000 designs, all registers are implemented in the macrocells. If your XC9000 design contains IFD symbols, CPLD software automatically implements them using macrocell registers. Also, the obsoleted XC7000 global attribute REG_OPT is ignored.
- In XC7000 software, the obsoleted “F” and “H” signal attributes were used to target Fast Function Blocks, High-density Function Blocks and FastInput pins (“F” attribute on an IBUF signals). In XC9000 designs, all macrocells, Function Blocks and device inputs with consistent timing across the device. When targeting XC9000 devices, the fitter ignores “F” and “H” signal attributes. Critical speed paths should instead be identified using Timespecs.
- Like XC7000, INIT=R and INIT=S attributes are used to specify the initial states of XC9000 registers. In XC7000 devices, registers are initialized when either the device is powered up or if the Master Reset pin is pulsed. In XC9000 devices, registers are initialized only at power-up. If you need to re-initialize XC9000 registers after power-up, you should use the asynchronous clear (CLR) or preset (PRE) pins on applicable library symbols. If you connect CLR or PRE pins to an ordinary device input (IBUF), the software automatically allocates the global set / reset (GSR) pin, if possible. Otherwise you can use the BUFGSR symbol instead of IBUF to explicitly use the GSR pin.
- In XC7000, the “preload optimization” feature allows the software to alter the initial states of flip-flops to achieve better logic optimization. In XC9000, initial state selection has no effect on logic optimization. All registers in XC9000 designs initialize to zero unless otherwise specified. Preload optimization is not

performed by CPLD software when targeting XC9500 devices. The obsoleted XC7000 global attribute PRELOAD_OPT is therefore ignored.

- In XC7000 designs, arithmetic logic is implemented using the built-in hardware carry chain. In the XC9000 schematic library, the same arithmetic macro symbols are implemented using ordinary combinational logic. The resulting implementation may require more macrocell resources and have longer delay paths than in XC7000.
- If you used PLUSASM-language equation files for behavioral modules (not recommended for new design entry), the XC7000-style arithmetic carry equations (output.ADD=VCC) are not supported by XC9000. You will need to redesign your arithmetic functions using conventional combinational logic.
- XC7000 devices support wire-ANDing of both device inputs and macrocell feedbacks within its UIM interconnect structure. XC9000 devices support wire-ANDing among only macrocell feedbacks in its FastCONNECT. If your design uses the OPT=UIM attribute on an AND or NAND gate to specify wire-ANDing, the CPLD software, when tergeting XC9000, obeys the OPT=UIM attribute only if the gate has no inputs taken directly from device pins. If one or more of the gate inputs connect to device inputs (IBUFs), the OPT=UIM attribute is ignored and the device inputs will be ANDed using macrocell logic; some or all of the macrocell feedbacks may be wire-ANDed in the FastCONNECT at the discretion of the software.

Schematic Conversion Procedure

When you re-target a design from a different Xilinx device family, you may need to perform a conversion procedure to replace the symbols in your existing schematic with symbols from the new CPLD library, depending on your design entry tool. Because of the Xilinx Unified Library, such conversions are automatic and produce functionally equivalent results with little or no manual changes required. The conversion is typically performed so that the macro symbols in your design schematic link to the underlying macro schematics in the new CPLD family's library rather than any other device library. Also, some primitive symbols being translated from another device library may be implemented as macro symbols in the new library, and vise-versa.

Before you convert your schematics to use a new CPLD library, you should remove any symbols that do not exist in the new library, typically replacing them with similar symbols from the new library. After you convert your schematic, any unsupported symbols may no longer be visible in your schematic, and this may make it more difficult to determine the appropriate replacement logic.

Note: If you are converting an FPGA design containing RAM, ROM, XBLOX symbols or other elements that do not have CPLD equivalents, you cannot retarget your design unless you redesign those portions.

Using Viewlogic on Workstations

In this example, we are converting a Viewlogic schematic originally implemented using the XC7000 library into a schematic targeting the XC9000 library.

1. Copy your existing schematic file(s) from the project directory used for the other device family (XC7000) into the project directory you want to use for the new target library (XC9000). For example:

   ```
   cp proj7000/sch/design1.1 proj9000/sch
   ```

2. Use the Viewlogic project management facility (or edit the viewdraw.ini file) in your XC9000 project directory to list both the new library (XC9000) and the other family's library (XC7000) in the Search Order. For example, if you are converting an XC7000 design to XC9000, your viewdraw.ini file would contain the following two lines:

   ```
   DIR [r] installation_path/unified/xc9000 (xc9000)
   DIR [r] installation_path/unified/xc7000 (xc7000)
   ```

3. Go to a system command window that is properly configured to run Viewlogic software. (The $path should include Viewlogic software and the $WDIR variable should be properly set.) Your current working directory should be your project directory containing the designs to be converted.

4. Invoke the Viewlogic `altran` utility to automatically replace all symbols in your design from the old library (XC7000) with

corresponding symbols from the new library (XC9000), as follows:

`altran -p` *design_name old_library***`=`***new_library*

where *old_library* is the library alias of the device family from which you are converting and *new_library* is the alias of your new target library. For example:

```
altran -p design1 xc7000=xc9000
```

Using Viewlogic PRO Series on PC

In this example, we are converting a Viewlogic schematic originally implemented using the XC7000 library into a schematic targeting the XC9000 library.

1. Start PROflow. If necessary, use the **`Project Manager`** to select the project you wish to convert. Your Viewlogic project only needs to be configured to access the Xilinx library from which you are converting (XC7000).
2. Use the Viewlogic `altran` conversion program to replace all symbols in your design from the old library (XC7000) with corresponding symbols from the new library (XC9000), as follows:
   a) Select the **`Design Entry`** button.

   b) Click on **`Select Family`**. A list of families will appear. Select **`XC9000`** and click on **`OK`**. The **`Altran`** menu will appear.

   c) Under **`Current Technology Aliases,`** select **`XC9000`** (the name of the family from which you are converting).

   d) Select **`XC9000`** under **`Target Technology Aliases`**, then select **`OK`**. The conversion program will execute.

Using OrCAD on PC

To retarget an OrCAD schematic, simply reconfigure OrCAD SDT to access the new target family's library. Change (cd) to your working directory and run `xdraft` specifying the new target family.

For example, if you are re-targeting a design for XC9000, type the following at the DOS prompt while in your working directory:

```
xdraft 9
```

The schematic itself requires no conversion processing. The next time you open your schematic, you will automatically be accessing the new target library.

Processing a Design After Conversion

After converting a schematic from a different device family, perform the following steps, as applicable:

1. Remove all attributes except INIT, FAST, and timing specifications. Make sure timespec syntax is up to date because EPLD software does not accept old timespec syntax (use syntax as described in this manual). Change the values of PART and LOC attributes as needed, or remove them.
2. On PC when you return to the Design Manager, create a new Xilinx project for the converted schematic design.

   From the Design Manager click on the **`File`** menu and select **`New Project`**.

   Enter a new project name to use for XC7000 or XC9000 implementation.

   From the **`Target Family`** select **`XC7300`** or **`XC9500`**.

   Before processing the design, open the Implementation Options menus and select the options available for the new device family.

   On workstations, simply run the **`xepld`** command using optional parameters that are appropriate for the new device family.
3. When you perform either functional or timing simulation, remember to pulse the PRLD signal High then Low. FPGA families use a GSR or GR signal for initialization.
4. If you wish to perform timing simulation, you may have to change the internal nodes you drive and monitor. The EPLD fitter optimizes the logic differently than FPGAs, which makes many of the internal nodes in the design invisible. However, all external signals are always visible.

Attribute Compatibility

The only schematic attributes common to FPGA devices and CPLD devices are:

- INIT=R | S

- Timing specifications for TIMESPEC and TIMEGRP symbols, including TNM
- FAST (output slew-rate control)
- FILE=*filename* for behavioral modules

The PART and LOC attributes are also used in a similar way by other families, but you must change their values when you change devices.

The following additional attributes are also common between the XC7000 and XC9000 CPLD families:

- LOWPWR=ON | OFF
- MINIM=ON | OFF
- OPT=OFF
- DEF = PLD

Any attributes contained in the converted design which are not supported by CPLD should be removed from the schematic before netlisting. If you converted from an FPGA family, make sure any timespec attributes are expressed using the FROM:TO syntax (described in Chapter 3) because CPLD software does not accept the older path-to-path timespec syntax.

Converting Behavioral Modules

If your design contains behavioral modules, you may need to perform some of these additional steps before running the fitter:

1. If your behavioral module contains state machine logic, you may need to change the encoding style of the state machines. You do not have to rewrite the logic, just the state assignment. For FPGAs, which are rich in registers, one-hot encoding using symbolic state representation is most efficient. For CPLDs, which are rich in product terms, binary encoding (or other encoding that minimizes state bits) is usually most efficient. Conversion may be unnecessary for very simple state machines.
2. Recompile your behavioral module specifying XC9000 or XC7000 as the target architecture. If you are using ABEL, specify Xilinx CPLD Netlist as the target output. ABEL will create a Plusasm-language equation file with extension .pld. In your schematic, continue to use the existing custom symbols, but change the

DEF=XABEL attributes to DEF=PLD. Also apply FILE=*filename* attributes as needed. To ensure that the software does not process old files, delete any *filename*.xnf files which may exist in your project directory.

3. If you are using a synthesis tool, recompile the behavioral module specifying XC9000 or XC7000 as the target technology library.

Chapter 3

Controlling Design Implementation

This chapter discusses the techniques for controlling how various parts of your design get implemented into a CPLD device. It concentrates mainly on the attributes you place in your schematic and the parameters you specify on the **xepld** command line.

This chapter describes how you can control the following aspects of design implementation:

- Device selection
- Register power-up state
- Macrocell power / speed trade-off
- Output slew rate
- Pin assignment and pin freezing
- Logic collapsing
- Timing specifications and optimization

Target Device Selection

By default, the fitter will automatically select an XC9000 family device for you, choosing, in general, the smallest part that will satisfy the needs and constraints of your design.

For Workstations

You can optionally specify a target device on the xepld command line when you run the fitter. The format of the part-type parameter on the xepld command line is:

xepld -p *ddddd-sspppp design_name*

where

ddddd is the device code, for example 95108

ss is the speed grade, for example 10

pppp is the package type and pin count, for example PC84

You may specify either a unique device code or a range of eligible devices from which the fitter will automatically choose. To specify a range of devices, you can use an asterisk (*) as a wildcard character. If you use an asterisk in the device code field, the string must begin with the number “9” or “7” (for example “9*-10PC84”). You may also specify an enumerated list of devices, separated by commas. If you use the asterisk character or an enumerated list in the xepld command, you must enclose the parameter string in quotes. For example, the following are valid part-type parameter specifications:

```
xepld -p 95108-10PC84 design1
xepld -p “95108-*PC84” design1
xepld -p “95108-10PC84,95108-7PQ*” design1
```

As an alternative, you can place the PART attribute in your schematic to select the target CPLD device for your design. Place the PART attribute on the schematic sheet itself (not on any particular component), as follows:

PART=*ddddd***-***sspppp*

You cannot, however, specify wildcards or lists in the schematic PART attribute as you can in the **xepld** command line. If you want the fitter to use the part specified in the schematic, you must include the parameter “-p indesign” on your **xepld** command line. For example:

```
xepld -p indesign design1
```

If you specify any part code in the **xepld -p** parameter (instead of “indesign”), it will override any PART attribute in your schematic.

Refer to the Release Document for a list of CPLD device codes supported by the current version of the fitter software.

For PCs

After you have opened a design, select **New Device** from the Implementation menu; the device selection menu appears, as shown in Figure 3-1.

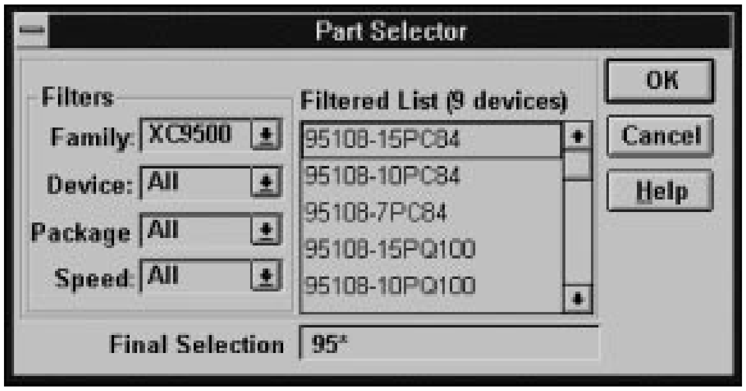


Figure 3-1 Device Selection Menu

From this menu, select either **`XC9500`** or **`XC7000`** as the family. Then specify the target device parameters that you want for your design. For example if you choose "**`All`**" for all Filters fields the software is free to choose any device in either the XC9500 or XC7000 families.

Note: You can also specify a part number in you design source file. This is described in the following sections.

Software Device Selection Criteria

If you select **`All`** for any device Filters field the software tries to fit your design within the range of possible devices you have selected by using the following selection criteria:

1. First, the software selects the smallest die.
2. Second, the software selects the smallest package.
3. Third, the software selects the slowest available device that meets your specified timing constraints.

The software will continue to try fitting your design into one of the possible range of selected devices until the first usable one is found or until there are no more devices to try.

Controlling Register Initial State

All registers in the CPLD device are initialized when the device is powered up. The initial state (preload value) of each register is programmable. Unless otherwise specified in your design, each register in an XC9000 design will initialize to the zero (reset) state at power-up. You can apply the INIT attribute to a registered component in your schematic to specify that it should initialize to the one (set) state as follows:

```
INIT=S
```

To specify that a component should preload to the zero (reset) state, use the INIT=R attribute (the default preload state for XC9000 designs).

The INIT attribute can be applied to flip-flops or any component containing a register, such as a counter macro.

Controlling Power Consumption

The power consumption of each macrocell in a CPLD device is programmable. The standard (default) setting consumes more power and produces shorter propagation delay. The low-power setting reduces power consumption for less speed-critical paths. By default, all macrocells in the design will operate in standard power mode.

Changing Power Mode for a Specific Component

You can apply the LOWPWR attribute to specific components in the schematic. To specify that the macrocell(s) used to implement a logic function are to operate in low-power mode, apply the following attribute to the corresponding component in your schematic:

```
LOWPWR=ON
```

The LOWPWR attribute can be applied to any logic or flip-flop component in the schematic, including a component that has multiple output signals. The LOWPWR attribute affects all macrocells used to implement the selected component.

If a component such as a logic gate or inverter is collapsed into another component, the LOWPWR attribute is not carried forward by the software. You may therefore need to apply the LOWPWR attribute to several components in a logic path to be sure that all macrocells used to implement the path are set to low-power mode.

The LOWPWR attribute has no effect on components that are not implemented using macrocell logic, such as I/O buffers.

Changing Global Power Mode on Workstation

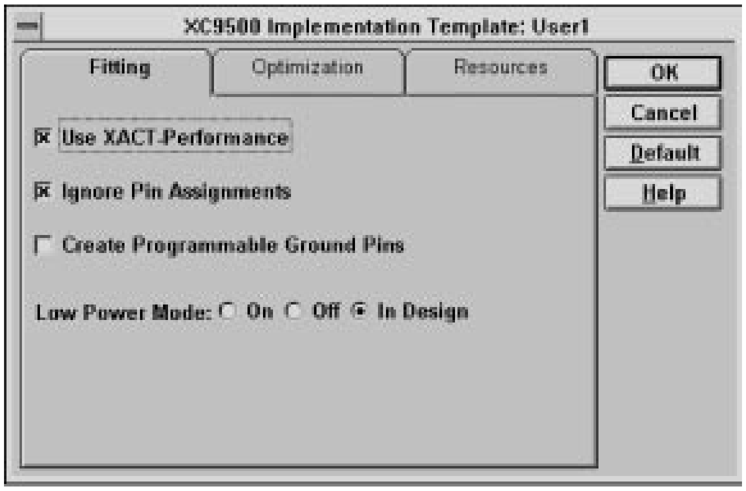
You can change the global power setting to use the low power mode throughout the design by specifying the "lowpwr" parameter on the xepld command line as follows:

`xepld -lowpwr` *design_name*

Changing Global Power Mode on PCs

To set all macrocells to the Low Power Mode throughout the design, set **`Low Power Mode`** to **`On`** in the **`Fitting`** menu of the **`Implementation Template`** in the Design Manager.

By setting the **`Low Power Mode`** to **`In Design`** in the template, macrocells will operate in standard power mode except where you specify the lowpwr attributes in the design.


XC9500 Implementation Template: User1

Fitting | Optimization | Resources

[✓] Use XACT-Performance

[✓] Ignore Pin Assignments

[ ] Create Programmable Ground Pins

Low Power Mode: On Off [✓ In Design]

Figure 3-2 Low Power Mode Set to In Design

Note: Low-power macrocells are slower than standard-power outputs. If you have a mixture of low- and standard-power macrocells, pay close attention to simulation results or the timing report to see how the power settings affect timing interactions.

Controlling Output Slew Rate

Each output of a CPLD device is programmable to operate either at full speed or with limited slew rate. Limiting the slew rate reduces output switching surges in the device. Slew rate control becomes important when your design uses a large number of outputs or you have transmission lines on your board which are sensitive to fast edge rates.

By default, all registers in a CPLD design have limited (slow) slew rate. If you want to disable the slew rate limitation of a device output to increase its switching speed, use the FAST attribute. Simply apply the following attribute to the OPAD (or IOPAD) symbol for each output to operate at full speed:

FAST

Controlling the Pinout

When you first run the fitter before your pinout is committed, the software automatically selects pin locations for your I/O signals. Pin locations are selected which will give you the greatest flexibility to iterate your design without having to move any of the pins. Each time the fitter successfully implements your design, it creates a guide file (*design_name*.gyd), which contains all the resulting pinout information. After you commit your pinout, subsequent design iterations cause the guide file to be read by the fitter and your committed pinout will be preserved.

We strongly recommend that you allow the software to automatically generate your initial pinout. Attempting to select your own initial pin preferences reduces the ability of the fitter to implement your design successfully the first time. It further reduces the amount of logic changes you could make after freezing your pinout.

Pin Freezing

If you have successfully fit a design into a CPLD device and you build a prototype containing the device, you will probably want to “freeze” the pinout.

Freezing Pins on a Workstation

The next time you iterate that design, you should specify the **`pinfreeze`** option on the **`xepld`** command line, as follows:

`xepld -pinfreeze` *design_name*

The **`-pinfreeze`** parameter tells the fitter to read and obey the pinout from the guide file that was saved the last time the fitter completed. The fitter will not move any of the pins contained in the guide file, even if it prevents the modified design from successfully mapping.

Freezing Pins on a PC

1. From the Design Manager, enter the Flow Engine. **`Tools`** → **`Flow Engine`**.
2. Select the down arrow adjacent to **`Guide Design`**. This will display a list of versions and revisions associated with the design.
3. Select the version and revision you want pinouts from and select **`OK`**. This will return you to the Flow Engine. When you **`RUN`** the design through, the pinouts from the selected revision will be used.

Guide Files

The pin locations stored in the guide file are specified based on the pad net names in the schematic. The pad nets are the nets that connect the IPADs to IBUFs and the OBUFs (or OBUFE or OBUFT) to OPADs (or IOPADs). If you change the label on any of the pad nets in your schematic, the pin will no longer be constrained to the location stored in the guide file. Renaming a pad net is a way you can relax the pinfreezing constraints on a design if your logic changes prevent the design from successfully fitting with your original pinout.

When you iterate your design while your pins are frozen, you are free to delete existing pins and/or add new pins to your schematic. The fitter will automatically select the best locations for any new pins you add, after placing all the existing pins constrained by the guide file.

Note: If you iterate your design and your pinout is not yet committed (you haven't built a prototype containing the device), you should not enable the pinfreeze option. Instead, allow the software to redefine the pinout of your modified design. This will continue to give you the greatest flexibility to iterate your design again after you commit your pinout.

Pin Assignment

You can assign explicit locations for pins in your design using the LOC attribute. To assign a pin location, apply the following attribute to a pad symbol (IPAD, OPAD or IOPAD) in your schematic:

LOC=*pin_name*

For PC and PQ type packages, the *pin_name* takes the form "P*nn*" where *nn* is a number. For example, for the PC84 package, the valid range for *pin_name* is P1 through P84. For grid array type packages (PG and BG), the *pin_name* takes the form "*rc*", where *r* is the row letter and *c* is the column number.

The LOC attribute cannot be applied to multi-bit pad components such as OPAD8. You must use individual pad symbols in your schematic if you want to perform pin assignment.

Whenever your design contains any LOC attributes, you should specify the target device type either using the **`xepld`** command's **`-p`** parameter or the schematic PART attribute (see Target Device Selection in this Chapter). LOC attributes are typically not compatible when retargeting a design between different package types, device types or device families.

LOC attributes are unconditional in that the software will not attempt to relocate a pin if it cannot achieve the specified assignment. If you specify a set of LOC attributes that the fitter cannot satisfy, the fitter will terminate with an error.

LOC attributes override the pin assignments in the guide file if you specify the pinfreeze option. This allows you to make explicit changes to your committed pinout. If you override an assignment in the guide file using LOC attributes, the software will issue a warning.

If your objective is to preserve a previously created pinout, we recommend you use the pinfreeze feature instead of back-annotating the existing pinout into your design schematic. The guide file saved from the previous design implementation contains additional information to help the fitter to successfully fit your modified design.

If your schematic contains LOC attributes but you want to let the fitter automatically assign all I/O pins, you can set the fitter to ignore all LOC attributes. This allows you to temporarily ignore all the LOC attributes in your schematic. This is useful if you want to test how your design fits a different target device without removing all the LOC attributes from your schematic.

Ignoring the LOC Attribute on a Workstation

On workstations you can specify the -ignoreloc parameter on the xepld command line:

`xepld -ignoreloc` *design_name*

Ignoring the LOC Attribute on a PC

Go to the Flow Engine and select:

`Setup → Options`

The Design Implementation Option menu appears. Select:

`Edit Template`

Then select the menu:

`Fitting`

Lastly, place a check in the box adjacent to **`Ignore Pin Assignments`**. Then click **`OK`**.

Pin Assignment Precautions

You can apply the LOC attribute to as many pad symbols in your design as you like. However, each pin assignment further constrains the software making it more difficult for the fitter to automatically allocate logic and I/O resources for the remaining I/O signals in your design.

When you manually assign output and I/O pins, you force the software to place associated logic functions into specific macrocells and specific function blocks. If the associated logic does not exceed the available function block resources (macrocells, product terms, and FastCONNECT inputs), the logic is mapped into the macrocell and the design will route in the FastCONNECT.

It is usually best to allow the fitter to automatically assign most or all of the pins based on the most efficient placement of logic in the device. The fitter automatically establishes a pinout which best allows for future design iterations without pin relocation. Any manual pin assignments you make in your design may not allow as much tolerance for changes in the logic associated with those pins, and in the logic physically mapped to nearby locations in the device.

If you are assigning pin locations to signals used as clocks, asynchronous set/reset, or output enable in your design, you should assign them to the GCK, GSR and GTS pins on the device if you want to take advantage of these global resources. The fitter will still automatically assign other clock, set/reset and output enable inputs to remaining GCK, GSR and GTS pins if available.

Controlling Logic Optimization

When you build combinational logic functions using simple gates and inverters, or when you use macros that contain gate-level logic paths, the software attempts to collapse as much of the logic as possible into the smallest number of CPLD macrocells. Any combinational logic function bounded between device I/O pins and flip-flops is subject to complete or partial collapsing. Collapsing the logic improves the speed of the logic path and can also reduce the amount of logic resources (macrocells, p-terms and FastCONNECT inputs) required to implement the function.

The process of collapsing logic into other logic functions is called “logic optimization”.

Collapsing Product Term Limit

When a larger combinational logic function consisting of several levels of AND-OR logic is completely collapsed (flattened), the number of product terms required to implement the function may grow considerably. By default, the fitter limits the number of p-terms used as a result of collapsing to 15 for XC9000 devices, and 17 for

XC7000 devices. If the collapsing of a logic level results in a logic function consisting of more than the p-term limit (after Boolean reduction), then the collapsing of that logic level is not performed and the function will be implemented using two or more levels of AND-OR logic.

Controlling Pterm Limits on a Workstation

The overall extent to which logic is collapsed throughout the design can be controlled using the “-pterms” parameter on the xepld command line:

`xepld -pterms` *nn design_name*

where *nn* is the maximum allowable number of p-terms that can be used to implement a logic function after collapsing. (The default *nn*=15 or 17.)

Controlling Pterm Limits on a PC

On PCs, controlling the Pterm limits is performed as follows:

1. Go to the Flow Engine and select:

   Setup → Options

2. The Design Implementation Option menu appears. Select:

   Edit Template

3. Then select the menu:

   Optimization

4. Lastly, place a value in the box adjacent to **Collapsing Pterm Limit**.

If the Path Delay is Not Satisfactory

If you find that the path delay of a larger, multi-level logic function is not satisfactory, try increasing the p-term limit parameter to allow the larger functions to be flattened further. For example, you may try increasing the p-term limit to 25 when rerunning the fitter.

The fitter report (*design_name*.rpt) indicates the number of p-terms used for each logic function. You should see these numbers increase as you raise the pterms limit, until the design is fully flattened. At the same time, you'll see the internal combinational nodes eliminated until none remain.

Note: Logic entered in equation form (such as through an ABEL behavioral module) remain at least as flat as originally expressed. That is, large sum-of-products equations are not broken down into smaller intermediate expressions to stay within the p-term limit parameter. Equations expressed using a larger number of p-terms than the global p-term limit remain intact and are implemented in one AND-OR logic level. Smaller combinational equations are collapsed into other equations up to the p-term limit, in the same manner as for logic gates in the schematic.

Preventing Collapsing of a Logic Node

Flattening typically increases the overall amount of p-term resources required to implement the design. Some designs which fit the target device initially may fail to fit if flattened too much. Other designs can be flattened completely and still fit. If you cannot increase the pterms parameter enough to sufficiently flatten a critical path and still fit the target device, you may try applying the logic optimization attribute to specific nodes in your design.

Applying the following attribute to a logic symbol in the middle of a logic function prevents collapsing of that symbol forward:

```
OPT=OFF
```

You can use OPT=OFF to break logic chains in non-speed-critical paths and prevent those functions from using too many p-terms. If you set the p-term limit parameter too high and your design no longer fits, try using OPT=OFF to reduce the size of selected non-critical paths.

The OPT=OFF attribute has no effect on any symbol that contains no macrocell logic, such as an I/O buffer.

When the OPT=OFF attribute is placed on a symbol, it inhibits logic optimization on all macrocells used to implement the symbol. For example, if you place OPT=OFF on a decoder symbol (like D2_4E), all outputs and internal nodes of the decoder will be prevented from collapsing.

If you want to prevent collapsing on a specific output signal from a macro symbol, you can place the OPT=OFF attribute on the signal (net) itself. When you place the OPT=OFF attribute on a signal, the fitter applies the attribute only to the primitive symbol that drives that signal.

Controlling Timing Paths

There are two mechanisms that can improve the timing of your design:

- Global Timing Optimization
- XACT Performance (Timespecs)

Timing Optimization

By default, the fitter performs global timing optimization on logic paths in your design. Timing optimization will shorten your critical paths as much as it can. In general, timing optimization optimizes logic and allocates the fastest available resources for the longest paths in your design, assuming all paths are equally critical. In some cases, the fitter trades off density for a speed advantage.

On workstations, if you do not want the fitter to perform timing optimization, you can specify the “-notiming” parameter on the xepld command line as follows:

`xepld -notiming` *design_name*

On PCs, to turn timing optimization off:

1. Go to the Flow Engine and select:

   `Setup → Options`

2. The Design Implementation Option menu appears. Select:

   `Edit Template`

3. Then select the menu:

   `Optimization`

4. Lastly, place a checkin the **`Off`** box adjacent to **`Timing`**.

XACT Performance

You can use Timing Specifications (T-Specs) to specify the maximum allowable delay between groups of components in your design. The software then optimizes and maps your design to achieve the timing defined by these specifications. This Xilinx CPLD timing-driven optimization is called XACT-Performance.

For schematic designs, T-Specs are specified in the schematic itself. Do not attempt to create or edit a constraints file (.cst) when using schematic design; the .cst file is overwritten when the schematic is read. Make all changes to the timing specifications within the schematic.

Timing Definitions

Delays and times are calculated as defined by the path types in this section.

The path types are defined as follows:

- Clock to Setup — Register to register cycle time, including clock to output and setup delay.

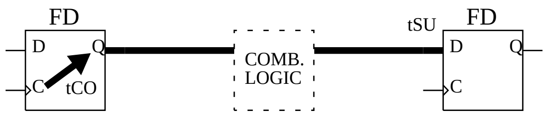


Figure 3-3 Clock to Setup Path

- Pad to Pad — Combinational pad to pad delay.

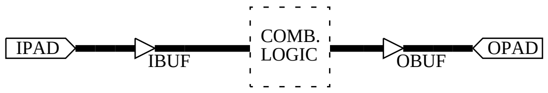


Figure 3-4 Pad to Pad Path

- Clock to Pad — Delay from the register clock input to the output pad.

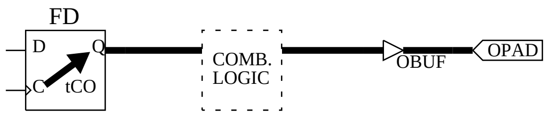


Figure 3-5 Clock to Pad Path

- Pad to Setup — Data path delay from the pad to the register data input. Includes the register setup time.

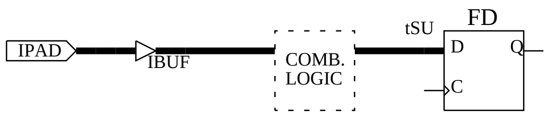


Figure 3-6 Pad to Setup Path

- Setup to Clock at the Pad — Setup time of data at the pad to clock at the pad. This path type includes only global clocks and product term clocks driven directly from input pads. If the data input is signal A and the clock input is signal CLK, the timing calculation for this type of path is as follows:

  Max(A to D) – Min(CLK to C)

  Max and Min are maximum and minimum propagation delays through the combinational logic.

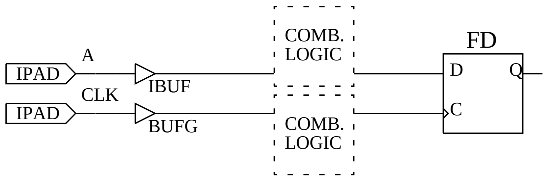


Figure 3-7 Setup to Clock at the Pad Path

- Clock Pad to Output Pad — Clock pad to output pad propagation delay. The clock can be a global or product term clock.

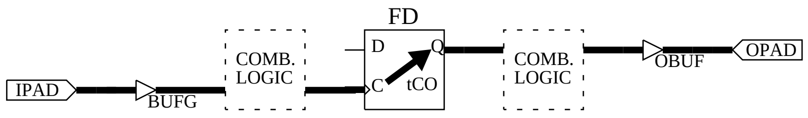


Figure 3-8 Clock Pad to Output Pad Path

- Paths Ending at Clock Pins of Flip-Flops — Delay from clock pad to register clock input. The clock can be a global or product term clock.

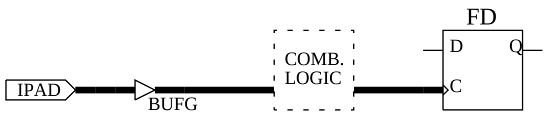


Figure 3-9 Path Ending at the Clock Pin of a Flip-Flop

The TIMESPEC Symbol

Timing specifications are placed on your schematic using a TIMESPEC primitive that contains the TIMESPEC Attribute Definitions which control the timing for paths between defined groups of components.

The TIMESPEC primitive, as illustrated in Figure 3-8, is 30 characters wide and contains TS attribute definitions. Each TIMESPEC primitive can hold up to eight TS attribute definitions. If you want to include more than eight TS attribute definitions, you can use multiple TIMESPEC primitives in your schematic.

Note: Though the TIMESPEC primitive is only 30 characters wide, you can create TS attribute definitions of any length by continuing on the next line.

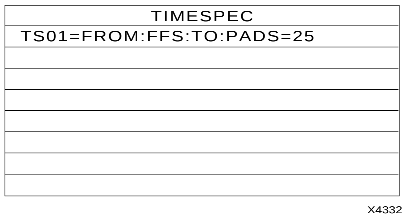


Figure 3-10 TIMESPEC Primitive

Defining Timing Path End Points

Specify the start and end points of your timing paths using one of the following methods:

- Refer to a predefined group by specifying one of the corresponding keywords — FFS, PADS.
- Create arbitrary groups within a predefined group by tagging symbols with TNM (pronounced *tee-name*) attributes.
- Create groups that are combinations of existing groups by using TIMEGRP symbols.
- Create groups by pattern matching on signal names.

Using Predefined Groups

You can refer to a group of flip-flops, input latches, or I/O pads, by using the corresponding keywords:

Table 3-1 Predefined Groups

Keyword	Group
FFS	Macrocell or IOB flip-flops
PADS	input/output pads

These predefined groups represent all symbols of that type. For example the following TS Attribute means that the delay between any two flip-flops must be no greater than 30 ns.

```
TS01=FROM:FFS:TO:FFS=30
```

And the following TS attribute means that the delay between any I/O pad and any flip-flop must be no greater than 25ns.

```
TS_OTHER=FROM:PADS:TO:FFS=25
```

Specifying Time Delay Units

Nanoseconds are the default units for specifying delay times in TS attributes. However, after specifying the maximum delay or minimum frequency numerically, you can enter the unit of measure by specifying the following:

- NS for nanoseconds
- MHZ for megahertz
- US for microseconds
- KHZ for kilohertz

The software converts all units to nanoseconds and rounds them to 0.1 ns accuracy.

Disabling Timing Specifications

If you have placed T-specs in your schematic but want to run the fitter without using your T-specs, you can temporarily disable XACT Performance. Specify the **`-ignorets`** parameter on the **`xepld`** command line as follows:

`xepld -ignorets` *`design_name`*

On a PC, disable timing specifications as follows:

1. Go to the Flow Engine and select:

   `Setup → Options`

2. The Design Implementation Option menu appears. Select:

   `Edit Template`

3. Then select the menu:

   `Fitting`

4. Lastly, remove the checkin the box adjacent to **`Use XACT-Performance`**.

Reducing Levels of Logic

The XC9000 and XC7000 architecture, like most CPLD devices, is organized as a large, variable-sized combinational logic resource (the AND-array and XOR gate) followed by a register. If you place combinational logic before a register in your design, the fitter maps the logic and register into the same macrocell. The output of the register is then directly available at an output pin of the device. If, however, you place logic between the output of a register and the device output pin, a separate macrocell must used to perform the logic, decreasing both the speed and density of your design. Figure 3-11 shows two functionally similar designs, one that is efficient for CPLD architectures and one that is inefficient.

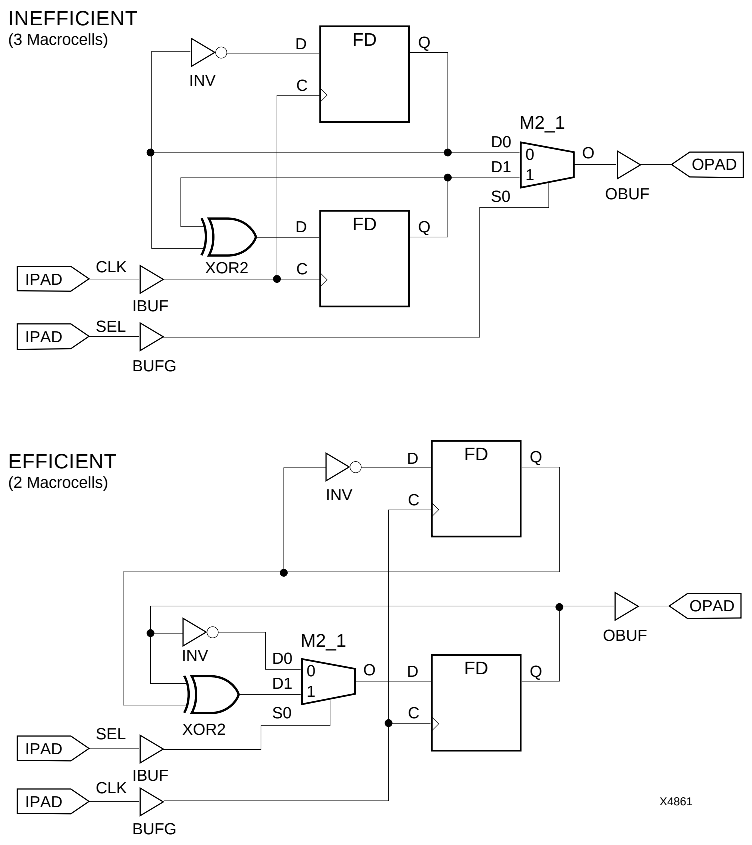


Figure 3-11 Reducing Levels of Logic

Chapter 4

Design Applications

This chapter describes some of the more useful techniques for expressing efficient CPLD designs. These examples are suggestions and guidelines only, and may not apply to your particular design.

Read-Back Registers

Figure 4-1 shows a simple read-back register. Data is written from the IOPAD to the register on the rising edge of the clock if READ_ENABLE is inactive and WRITE_ENABLE is active. Data is read from the IOPAD when READ_ENABLE is active.

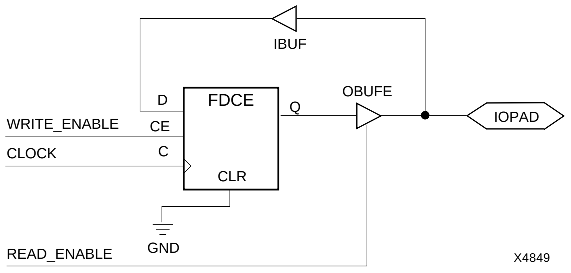


Figure 4-1 Read-Back Register Example

Bidirectional Signals and Buses

Figure 4-2A shows how to specify a bidirectional pin. Figure 4-2B shows that you can have a bidirectional signal passing through the chip. To make a bidirectional bus, use bus components as shown in Figure 4-2C.

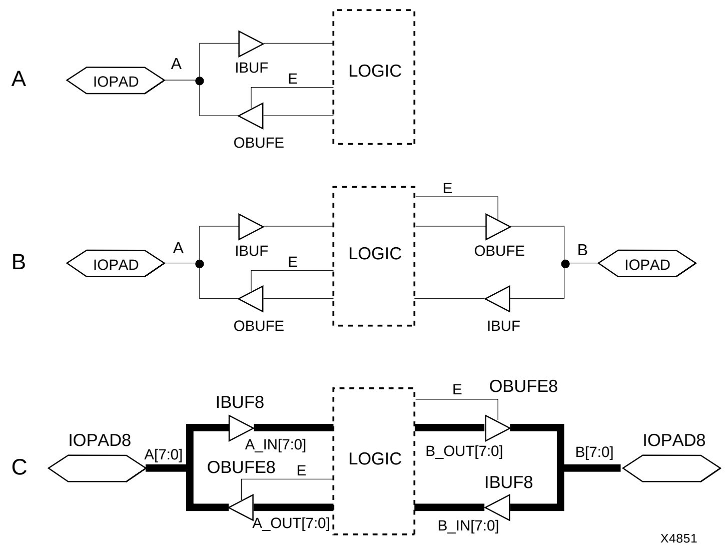


Figure 4-2 Bidirectional Signals and Buses

Multiplexing 3-State Signals

Three methods of multiplexing 3-state signals are shown in Figure 4-3 on the next page. Which method you choose depends on your application, resources, and speed requirements, although method C, which uses a multiplexer, is usually best for CPLD designs.

Method A, shown in Figure 4-3A, uses 3-state buffers instead of a multiplexer. The advantage of method A over method C is that method A uses only one Function Block input in the macrocell that sends the signal off-chip. The disadvantage of method A is that macrocell feedback is lost because the outputs are 3-stated; therefore counters will not work with Method A, but will work with Method C.

Method B, shown in Figure 4-3B, requires that you tie the signals together off-chip. This method results in a short clock-to-out delay and uses fewer macrocells than methods A and C. However, it uses more pins than method A or C.

Method C, shown in Figure 4-3C, uses a multiplexer instead of 3-state buffers. This method results in a longer clock-to-out delay than method B, although you can shorten this delay to that of method B by registering the output of the multiplexer and asserting the select signals one clock cycle in advance. This method uses more macrocells than method B, but uses fewer pins.

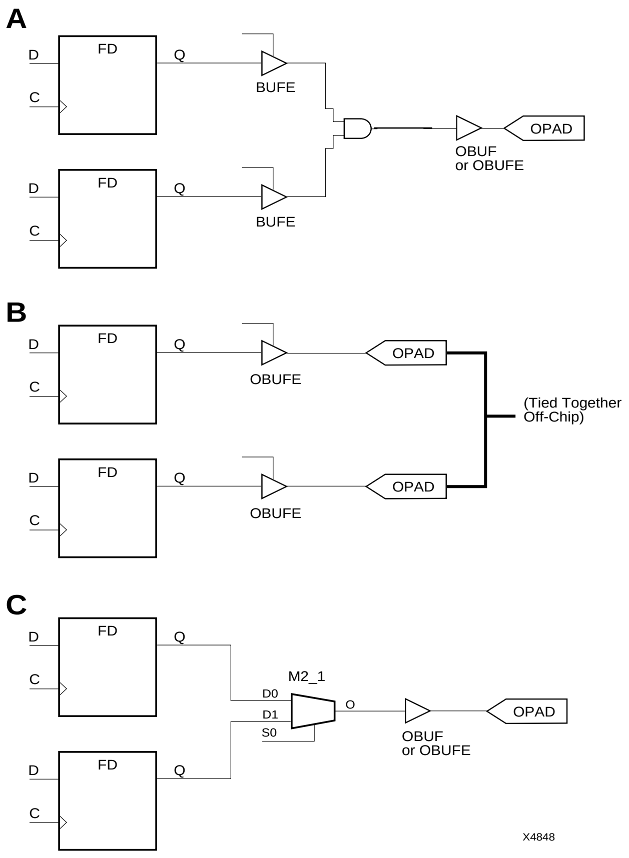


Figure 4-3 Methods of Multiplexing 3-State Signals

Combinational Feedback Loops

The simple expression of a D-type latch contains inherent logic hazards which could result in unpredictable results when run through the fitter.

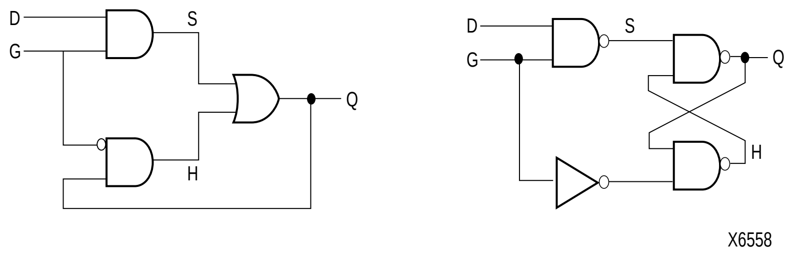


Figure 4-4 Simple Mux and Cross-Coupled-NAND Latches

A timing malfunction can occur if the logic is divided between two separate macrocells by the fitter. Figure 4-5 illustrates what can happen.

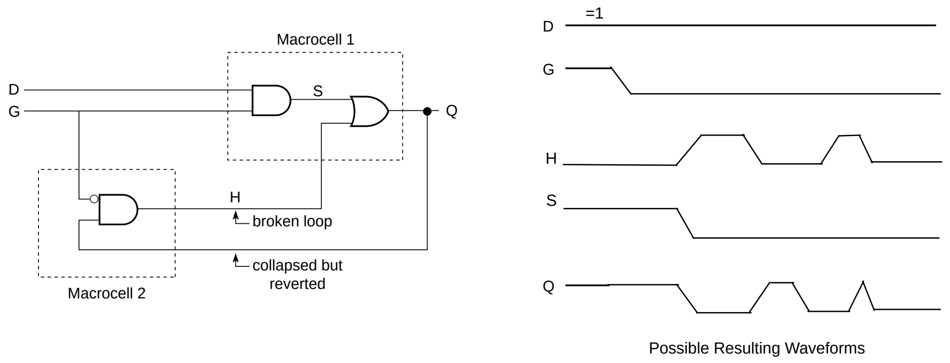


Figure 4-5 Malfunction of Physical Implementation

If you implement the D-type latches with proper redundant logic, the problem will not occur. Figure 4-6 shows two solutions for schematic implementation of D-type latches.

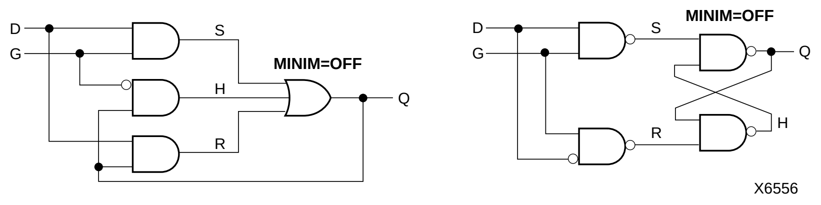


Figure 4-6 D-type Latch Solutions

When you create redundant logic in a schematic, remember to specify the MINIM=OFF attribute on the final output gate to prevent the software’s Boolean minimization routine from removing the redundant logic

Appendix A

Common Questions and Answers

This appendix lists frequently asked questions about EPLD software and its CAE tool interfaces, and gives explanations and solutions.

Drawing the Design

This section lists problems you may encounter because your CAE tool drawing package is not properly configured for XEPLD software.

Why Do I See White Boxes Instead of Symbols?

If you are a Viewlogic user and your schematic contains symbols from a device family library (such as XC9000) that is not included in your viewdraw.ini file, you see white boxes when you view your schematic.

A likely cause of this problem is forgetting to run the Altran program when converting from one device family to another; see the “Device-Independent Design” chapter for details. Even after you run Altran, components from the old library that are not in the new library appear as white boxes — you should find equivalent components that are compatible with the new library.

Another likely cause is not configuring viewdraw.ini properly, with correct path names and library aliases. The example in the “Getting Started with Schematic Design” chapter includes information about how to configure Viewlogic software for the XEPLD device families.

Fitting the Design

This section lists problems you may encounter when you fit the design.

What Does “Unrecognized Symbol” Mean?

If you get this error message:

```
xr55: [Warning]‘ADECODE is an unrecognized symbol
for the EPLD family. If this symbol is a
behavioral module, make sure you have added
FILE=module and DEF=PLD attributes to the symbol.
If the symbol is a custom schematic component,
check that a schematic exists for it. If the
symbol is a standard library component, make sure
the target EPLD family supports it. If you are
using a synthesis tool, resynthesize th logic
targeting an EPLD family.
```

it means one of the following occurred:

- You did not properly link a behavioral design to a schematic symbol. You need to add attributes FILE=*filename* and DEF=PLD to the schematic symbol.
- If the symbol is a user macro, check that a schematic exists for it and that an .xnf file was generated.
- If the symbol is a Xilinx library component, make sure the target EPLD family supports it.
- If you are using a synthesis tool, recompile the logic targeting an EPLD family.

Simulating the Design

This section lists problems you may encounter during functional or timing simulation.

Why Are My Registers Stuck at the Preload Value?

At the beginning of a simulation, you must pulse the PRLD signal High then Low.

The assertion of the pulse forces all registers to a known state. The de-assertion allows the registers to change states. If you do not de-assert PRLD, your registers cannot change state.

Why Are My Internal Nodes Not Visible During Timing Simulation?

The EPLD fitter optimizes your design for efficiency, eliminating many internal combinatorial nodes. (If you are a Viewlogic user and you view your back-annotated schematic during timing simulation, these nodes appear with a “?”.)

Nets between IPADs and IBUFs are observable, as are nets between OBUFs and OPADs. Other nets may or may not be observable, depending on the results of optimization.

Appendix B

Attributes

This appendix describes all of the attributes that you can place into your schematic for a CPLD design. The attributes supported for CPLD designs are as follows:

- DEF=PLD
- FAST
- FILE=*filename*
- INIT={R | S}
- LOC=*pin_name*
- LOWPWR=ON
- MINIM=OFF
- OPT=OFF
- PART=*part_type*
- TNM=*time_group*
- TS*nn*=*time_spec*

Target Device Selection Attribute — PART

You can place the global PART attribute in your schematic to select the target device for your design. Refer to the Release Document for a list of CPLD device names supported by the software.

Selecting a part type other than “indesign” in the -p parameter of the xepld command line overrides any PART attribute in your schematic.

Note: The fitter will automatically select a part for you, choosing, in general, the smallest part that will satisfy the needs and constraints of your design.

The format of the PART value is as follows:

PART=*dddd*-*sspppp*

dddd is the device number, for example 95108

ss is the speed grade, for example 10

pppp is the package type and pin count, for example PC84

Viewlogic Procedure

Apply the PART attribute as an unattached schematic attribute. Follow these steps:

1. Deselect all components by clicking on a blank area of your schematic.
2. Use the **Add** → **Attribute** command and enter the PART attribute string and press enter.
3. Position the attribute anywhere in the schematic and click with the left mouse button.

Behavioral Module Attributes — FILE and DEF

The FILE=*file_name* attribute on a custom symbol specifies the name of the file containing the logical definition for that symbol when the logic is expressed in behavioral form instead of an underlying schematic. If the logic for your custom symbol is defined by an underlying schematic (i.e., a user macro), you need no FILE or DEF attributes.

Specify the directory path if necessary. Do not specify the file extension in the FILE=*file_name* attribute.

The DEF=PLD attribute is used in addition to the FILE=*file_name* attribute when the file defining the symbol's logic is in the Plusasm equation format. Plusasm-formatted files are typically produced by PLD compiler tools such as ABEL. If *file_name* is an XNF-formatted netlist file, you need no DEF attribute. XNF-formatted files are typically produced by logic synthesis tools such as Synopsys.

Pin Assignment Attribute — LOC

Use the LOC=*pin_name* attribute on a PAD symbol to assign the signal to a specific device pin. The PAD symbols are IPAD, OPAD, IOPAD, and UPAD. The pin name is P*nn* for PC packages; the *nn* is a pin number. The pin name is *rc* (row, column) for PG packages. Examples are LOC=P24 and LOC=G2.

Pin assignments are unconditional in that the software will not attempt to relocate a pin if it cannot achieve the specified assignment. You can apply the LOC attribute to as many PADs in your design as you like. However, each pin assignment further constrains the software as it automatically allocates logic and I/O resources to internal nodes and I/O pins with no LOC attributes.

Note: Pin assignment using the LOC attribute is not supported for bus components such as OBUF8.

Power Setting Attribute — LOWPWR

By default, all macrocells operate in the standard power mode, providing the fastest possible speed. If you specify the “lowpwr” parameter on the xepld command line, all macrocells operate in low-power mode.

To specify low-power mode for macrocells used by an individual symbol, use the LOWPWR=ON attribute in the schematic.

The LOWPWR attribute is ignored if assigned to a symbol that uses no macrocells, such as an inverter or an I/O buffer.

Logic Optimization Attribute — OPT=OFF

Use the logic optimization attributes to control collapsing at specific points in your design. Logic optimization attributes are normally not required to process CPLD designs.

The OPT=OFF component attribute inhibits collapsing of all macro-cells used by a symbol.

The logic optimizer normally collapses the levels of logic to remove intermediate nodes. Components are optimized forward into components connected to their outputs. If you build combinational logic using low-level gates and macros containing gates, the software attempts to pack all logic bounded between device I/O pins and registers into a single macrocell.

The OPT attribute has no effect on any symbol that contains no macrocell logic, such as an I/O buffer.

The OPT=OFF attribute can be used to prevent optimization if it appears that the software is collapsing logic in a way that prevents a successful fit.

Register Preload State — INIT

The INIT attribute specifies the initialization value to be preloaded into a register upon power-up. INIT=R specifies a preload value of 0 (Reset) and INIT=S specifies a preload value of 1 (Set). The default for all registers in a CPLD design is 0 (reset). This attribute can be applied to flip-flops or any component containing a register.

Output Slew Rate — FAST

The FAST attribute can be placed on an OPAD (output pad) or IOPAD symbol (primitive or macro) to select the fast slew-rate operation of the corresponding CPLD output-pin driver. The default is reduced (slower) slew-rate, which reduces output switching surges in the device.

Minimization of Redundant Logic — MINIM

The MINIM=OFF attribute tells the fitter to disable Boolean logic minimization for the attached component. You need to use the MINIM=OFF attribute if you want to specify redundant logic in a portion of your design to avoid a potential race condition; for example, you would use MINIM=OFF on the output gate when designing combinational feedback loops and latches.

Timing Specification Attributes — TSnn and TNM

The T-spec attribute definitions specify the maximum delay between groups of components. They begin with the letters “TS” and a unique identifier that can consist of letters, numbers, or the underscore character (_). The value of the T-spec attribute consists of a FROM-TO expression specifying the timing requirements between specific end points. The full syntax is shown as follows:

TS*identifier***=FROM:***group1***:TO:***group2***=***delay*

The parameters *group1* and *group2* can be any of the following:

- Predefined groups consisting of FFS or PADS which are discussed in the “Using Predefined Groups” section.
- Previously created TNM identifiers which are introduced in the “Creating Arbitrary Groups Using TNMs” section.
- Groups defined in TIMEGRP symbols which are introduced in the “Creating New Groups from Existing Groups” section.

The *delay* parameter defines the maximum delay for the attribute, using nanoseconds as the default unit of measurement. Other units of measurement such as MHZ may also be used.

The format of the TNM attribute is:

TNM=*group_name*

Note: Keywords, such as FROM and TO, appear in this document in upper case; however, you can enter them in the TIMESPEC primitive in either upper or lower case.

The following examples show typical TIMESPEC attribute definitions:

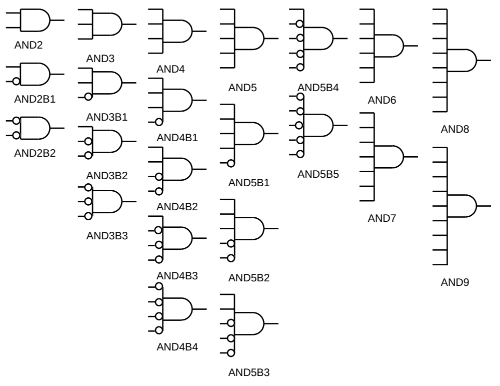
```
TS01=FROM:FFS:TO:FFS=30
TS_OTHER=FROM:PADS:TO:FFS=25
```

Appendix C

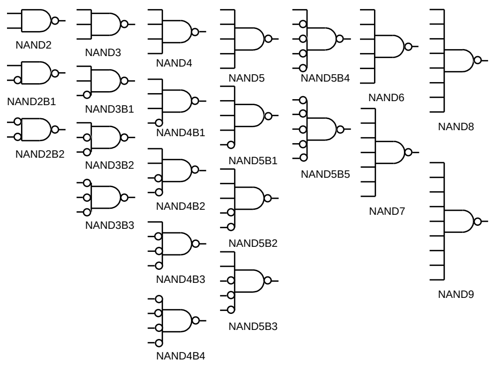
XEPLD Library Selection Guide

Logic Gates

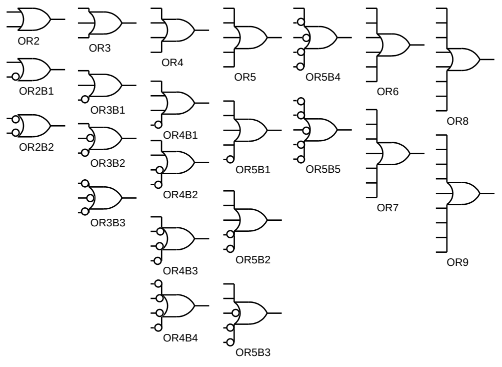
ANDs*



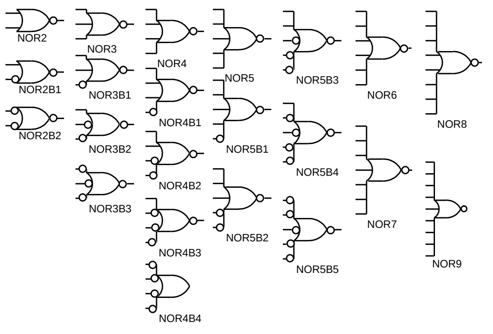
NANDs*



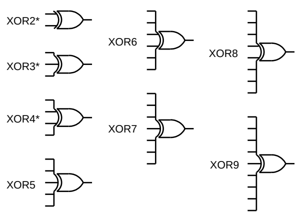
ORs*



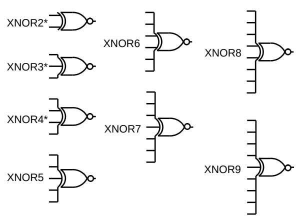
NORs*



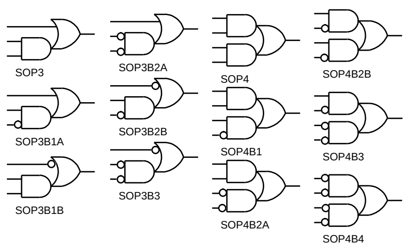
XORs



XNORs



Sums of Products



Component Name	Description/Features
Buffers and Inverters	
BUF*, BUF4, BUF8, BUF16	Non-inverting buffer
BUFE, BUFE4, BUFE8, BUFE16	Internal 3-state buffer with active-high enable
BUFG*	Global clock input buffer
BUFGSR*	Global asynchronous set/reset input buffer
BUFGTS*	Global 3-state control input buffer
BUFT*, BUFT4, BUFT8, BUFT16	Internal 3-state buffer with active-low enable
INV*, INV4, INV8, INV16	Inverter
Flip-Flops	
FD, FD4, FD8, FD16	D flip-flop
FDC	D flip-flop with async. clear
FDCE, FD4CE, FD8CE, FD16CE	D flip-flop with clock enable, async. clear
FDCP*	D flip-flop with async. preset, async. clear
FDCPE	D flip-flop with clock enable, async. preset and clear
FDP	D flip-flop with async. preset
FDPE	D flip-flop with clock enable, async. preset
FDR	D flip-flop with sync. reset
FDRE, FD4RE, FD8RE, FD16RE	D flip-flop with clock enable, sync. reset
FDRS	D flip-flop with sync. reset, sync. set
FDRSE	D flip-flop with clock enable, sync. reset and set
FDS	D flip-flop with sync. set
FDSE	D flip-flop with clock enable, sync. set
FDSR	D flip-flop with sync. set and reset
FDSRE	D flip-flop with clock enable, sync. set and reset
FJKC	J-K flip-flop with async. clear
FJKCE	J-K flip-flop with clock enable, async. clear
FJKCP	J-K flip-flop with async. clear and preset
FJKCPE	J-K flip-flop with clock enable, async. clear and preset
FJKP	J-K flip-flop with async. preset
FJKPE	J-K flip-flop with clock enable, async. preset
FJKRSE	J-K flip-flop with clock enable, sync. reset and set
FJKSRE	J-K flip-flop with clock enable, sync. set and reset
FTC	Toggle flip-flop with async. clear
FTCE	Toggle flip-flop with clock enable, async. clear
FTCLE	Loadable toggle flip-flop with clock enable, async. clear
FTCP	Toggle flip-flop with async. clear and preset
FTCPE	Toggle flip-flop with clock enable, async. clear and preset
FTCPLE	Loadable toggle flip-flop w/ clock enable, async. clear & preset
FTP	Toggle flip-flop with async. preset
FTPE	Toggle flip-flop with clock enable, async. preset
FTPLE	Loadable toggle flip-flop with clock enable, async. preset
FTRSE	Toggle flip-flop with clock enable, sync. reset and set
FTRSLE	Loadable toggle flip-flop with clock enable, sync. reset and set

Component Name	Description/Features
FTSRE	Toggle flip-flop with clock enable, sync. set and reset
FTSRLE	Loadable toggle flip-flop with clock enable, sync. set and reset
X74_174	6-bit data register with asynchronous clear
X74_273	8-bit data register with asynchronous clear
X74_377	8-bit data register with clock enable
Latches	
LD, LD4, LD8, LD16	Transparent data latch
Shifters	
BRLSHFT4	4-bit barrel shifter
BRLSHFT8	8-bit barrel shifter
SR4CE, SR8CE, SR16CE	Shift register with clock enable, async. clear
SR4CLE, SR8CLE, SR16CLE	Loadable shift register with clock enable, async. clear
SR4CLED, SR8CLED, SR16CLED	Loadable left/right shift register with clock enable, async. clear
SR4RE, SR8RE, SR16RE	Shift register with clock enable, sync. reset
SR4RLE, SR8RLE, SR16RLE	Loadable shift register with clock enable, sync. reset
SR4RLED, SR8RLED, SR16RLED	Loadable left/right shift register with clock enable, sync. reset
X74_164	8-bit serial-in parallel-out shift register with async. clear
X74_165S	8-bit loadable serial/parallel-in parallel-out shift register with clock enable
X74_194	4-bit loadable left/right serial/parallel-in parallel-out shift register
X74_195	4-bit loadable serial/parallel-in parallel-out shift register
Counters	
CB2CE, CB4CE, CB8CE, CB16CE	Cascadable binary counter with clock enable, async. clear
CB2CLE, CB4CLE, CB8CLE, CB16CLE	Loadable cascadable binary counter with clock enable, async. clear
CB2CLED, CB4CLED, CB8CLED, CB16CLED	Loadable up/down binary counter with clock enable, async. clear
CB2RE, CB4RE, CB8RE, CB16RE	Cascadable binary counter with clock enable, sync. reset
CB2RLE, CB4RLE, CB8RLE, CB16RLE	Loadable cascadable binary counter with clock enable, sync. reset
CB2X1, CB4X1, CB8X1, CB16X1	Loadable cascadable up/down binary counter with async. clear
CB2X2, CB4X2, CB8X2, CB16X2	Loadable cascadable up/down binary counter with sync. reset
CD4CE	4-bit cascadable BCD counter with clock enable, async. clear
CD4CLE	4-bit loadable cascadable BCD counter with clock enable, async. clear
CD4RE	4-bit cascadable BCD counter with clock enable, sync. reset
CD4RLE	4-bit loadable cascadable BCD counter with clock enable, sync. reset
CJ4CE, CJ5CE, CJ8CE	Johnson counter with clock enable, async. clear
CJ4RE, CJ5RE, CJ8RE	Johnson counter with clock enable, sync. reset
CR8CE, CR16CE	Negative-edge binary ripple counter with clock enable, async. clear
X74_160	4-bit loadable cascadable BCD counter with parallel/trickle enables, async. clear
X74_161	4-bit loadable cascadable binary counter with parallel/trickle enables, async. clear
X74_162	4-bit loadable cascadable BCD counter with parallel/trickle enables, sync. reset

Component Name	Description/Features
X74_163	4-bit loadable cascadable binary counter with parallel/trickle enables, sync. reset
X74_168	4-bit loadable cascadable up/down BCD counter with parallel/trickle enables
X74_390	4-bit BCD/bi-quinary ripple counter with negative-edge clocks, async. clear
Multiplexers	
M2_1	2-to-1 multiplexer
M2_1B1	2-to-1 multiplexer with D0 inverted
M2_1B2	2-to-1 multiplexer with D0 and D1 inverted
M2_1E	2-to-1 multiplexer with enable
M4_1E	4-to-1 multiplexer with enable
M8_1E	8-to-1 multiplexer with enable
M16_1E	16-to-1 multiplexer with enable
X74_150	16-to-1 inverting multiplexer with enable
X74_151	8-to-1 multiplexer with enable and complementary outputs
X74_152	8-to-1 inverting multiplexer
X74_153	Dual 4-to-1 multiplexer with enables
X74_157	Quad 2-to-1 multiplexer with enable
X74_158	Quad 2-to-1 inverting multiplexer with enable
X74_298	Quad 2-input multiplexers with storage, negative-edge clock
X74_352	Dual 4-to-1 inverting multiplexer with enables
Decoders	
D2_4E	2- to 4-line decoder/demultiplexer with enable
D3_8E	3- to 8-line decoder/demultiplexer with enable
D4_16E	4- to 16-line decoder/demultiplexer with enable
X74_42	4- to 10-line active-low BCD-to-decimal decoder
X74_138	3- to 8-line active-low decoder/demultiplexer with enables
X74_139	2- to 4-line active-low decoder/demultiplexer with enable
X74_154	4- to 16-line active-low decoder/demultiplexer with enables
Encoders	
X74_147	10- to 4-line active-low priority encoder
X74_148	8- to 3-line cascadable active-low priority encoder
Comparators	
COMP2, COMP4, COMP8, COMP16	Identity comparator
COMPM2, COMPM4, COMPM8, COMPM16	Magnitude comparator
X74_L85	4-bit expandable magnitude comparator
X74_518	8-bit identity comparator with enable
X74_521	8-bit active-low identity comparator with enable
Arithmetic Functions	
ACC1, ACC4, ACC8, ACC16	Loadable add/subtract accumulator
ADD1, ADD4, ADD8, ADD16	Adder
ADSU1, ADSU4, ADSU8, ADSU16	Adder/subtracter

Component Name	Description/Features
X74_280	9-bit odd/even parity checker/generator
X74_283	4-bit full adder with carry-in and carry-out
Input/Output Functions	
IBUF*, IBUF4, IBUF8, IBUF16	Input buffer
IOPAD*, IOPAD4, IOPAD8, IOPAD16	Input/output pad
IPAD*, IPAD4, IPAD8, IPAD16	Input pad
OBUF*, OBUF4, OBUF8, OBUF16	Output buffer
OBUFE, OBUFE4, OBUFE8, OBUFE16	3-state output buffer with active-high enable
OBUFT*, OBUFT4, OBUFT8, OBUFT16	3-state output buffer with active-low enable
OPAD*, OPAD4, OPAD8, OPAD16	Output pad
UPAD*	Unbonded I/O pad
Miscellaneous	
GND*	Ground-connection signal tag
VCC*	VCC-connection signal tag
TIMEGRP*	Timing specification group table
TIMESPEC*	Timing requirement specification table
TBLOCK	Title block annotation symbol
ASHEETL, ASHEETP, BSHEETL, CSHEETL, CSHEETP, DSHEETL, ESHEETL, ESHEETP	Sheet border annotation symbol

* XC9000 primitive symbol

Appendix D

Design Implementation and Simulation

This appendix describes how to invoke the CPLD fitter, and the commands used to prepare functional and timing simulation models.

Running the Fitter

From a Workstation

The **`xepld`** command invokes the CPLD design implementation software (the fitter). The command is run in a UNIX command window. Your current working directory must be set to the project directory which contains your design source files before invoking xepld. For Viewlogic designs, your project directory is the working directory containing the sch sub-directory in which your schematic file (*design_name*.1) is stored.

The format of the xepld command is:

`xepld [`*options***`]`** *design_name*

Invoking the **`xepld`** command with no parameters produces a listing of all available command-line options.

The *design_name* is the name of the top-level schematic file, without path qualifiers, and either with or without extension.

Viewlogic schematic files (sch / *design_name*.1) are supported and read directly by the **`xepld`** command. Schematics from other tools must first be translated into either an XNF-formatted netlist (*design_name*.xnf) or an EDIF-formatted netlist (*design_name*.edif).

If *design_name* is specified without extension, the **`xepld`** command searches for source files in the following order:

1. Viewlogic schematic (sch / *design_name*.1)
2. EDIF netlist (*design_name*.edif)

3. XNF netlist (*design_name*.xnf)

XEPLD Command Parameters

This section describes parameters that can be entered when using the **`xepld`** command on a workstation. For equivalent implementation options on a PC, see **Design Optimization Control** in Chapter 3 of the *XEPLD Reference Guide***.**

The [*options*] field of the xepld command represents an optional list of one or more command-line parameters. Invoking the xepld command with just the design name and no option parameters runs the fitter with all default conditions, including automatic device selection.

The following are the xepld command-line parameters that apply to schematic design entry:

- **-detail** — produces a detailed path timing report (*design_name*.tim) instead of the default summary report.
- **-ignoreloc** — temporarily ignores all LOC attributes in the schematic, allowing the fitter to assign the locations of all I/O pins.
- **-ignorets** — temporarily ignores all timing specification attributes in the schematic.
- **-lowpwr** — uses the low-power mode for all macrocells in the design (default is standard power except where LOWPWR=ON is specified in the schematic).
- **-mrinput**— allows the Master Reset pin to be used as an ordinary input (XC7000 only)
- **-nogck** — prevents the fitter from optimizing IBUF inputs used as clocks onto the device’s global clock input pins (GCK); only BUFG inputs are mapped to global clock pins.
- **-nogts** — prevents the fitter from optimizing IBUF inputs used for 3-state output enable control onto the device’s global 3-state control input pins (GTS or FOE); only BUFGTS or BUFFOE inputs are mapped to GTS pins.
- **-noifd**— prevents the fitter from optimizing simple flip-flops (FD) connected to device inputs (IBUF) onto the device’s input-pad registers (IFD); only IFD symbols in the schematic will use input-

pad registers (XC7000 only).

- **-nota** — bypasses the timing analyzer so that no static timing report is generated.
- **-notiming** — inhibits the default global timing optimization performed by the fitter; only paths with T-specs specified in the schematic are optimized to improve timing.
- **-p** *part_type* — specifies the target device type or set of devices from which to choose (default is automatic device selection); where *part_type* can be:
    - "*ddddd-sspppp*" — where *ddddd* is the device code (such as 95108), *ss* is the speed grade, *pppp* is the package code (such as PQ160), and an asterisk (*) can be used as a wildcard string (quotes required around *part_type* when asterisk is used).
    - "*ddddd-sspppp,ddddd-sspppp ...*" — a list of valid part-type specifications as defined above (quotes required).
    - indesign — signifies that the target device is specified by a PART attribute in the schematic.
- **-pinfreeze** — uses the guide file (*design_name*.gyd) from the last successful invocation of the fitter to reproduce the same pin locations (default is automatic pin assignment).
- **-pterms** *nn* — sets the limit to *nn* for the number of product terms allowed as a result of collapsing (default=15 for XC9000 and 17 for XC7000).
- **-s** *signature* — specifies the user signature string to be programmed into the device for identification purposes (default is the design name).

From a PC

1. From the Design Manager select the schematic file you want to process.

   File → Open Project

Select a file from the template's list or use the **Browse** key to search your directories for the file you want to process. If the file is listed on the template, highlight the file and click once on **Open**.

2. Go to the Flow Engine and select options:

   Tools → Flow Engine

   Setup → Options

3. The Design Implementation Option menu appears. Select:

   Edit Template

4. Then select from the three templates all the options you want to use and press **OK**.

5. To run the fitter, click once on the **run** key found in the Flow Engine.

Generated Reports

By default the fitter produces the following significant output files:

- Fitter report *(design_name*.rpt) — lists summary and detailed information about the logic and I/O pin resources used by the design, including the pinout, error and warning messages, and Boolean equations representing the implemented logic.
- Static timing report *(design_name*.tim) — shows a summary report of worst-case timing for all paths in the design; optionally includes a complete listing of all delays on each individual path in the design.
- Guide file *(design_name*.gyd) — contains all resulting pinout information required to reproduce the current pinout if the "pinfreeze" option is specified during the next invocation of the xepld command for the same design name. (The Guide file is written only upon successful completion of the fitter.)
- Programming file *(design_name*.jed) — is a JEDEC-formatted programming file to be down-loaded into the XEPLD device.
- Timing simulation netlist *(design_name*_tim.xnf) — an XNF-formatted netlist representing the implemented logic of the design, including all delays, consisting of XEPLD simulation model primitives.

Whenever the **xepld** command is invoked from a workstation, it copies any existing fitter report file (.rpt), timing report file (.tim), guide file (.gyd) and programming file (.jed) to the backup directory.

Functional and Timing Simulation

This section describes how to prepare a simulation model file for functional and timing simulation on the Workstation and PC environment.

Simulation on a Workstation

The **`xepldsim`** command prepares a simulation model file for either functional or timing simulation for one of the supported simulator tools. For functional simulation, the **`xepldsim`** command reads your source schematic design, plus any behavioral modules used in your design, and produces a simulation output file. For timing simulation, the **`xepldsim`** command translates the timing simulation netlist file (*design_name*_tim.xnf) produced by the xepld command into the required simulation output file(s).

The **`xepldsim`** command is run in a UNIX command window. Your current working directory must be set to the project directory which contains your design source files before invoking **`xepldsim`**.

The format of the **`xepldsim`** command is:

`xepldsim -`*target design_name*

Invoking the **`xepldsim`** command with no parameters produces a listing of all available command-line options.

The *design_name* is the name of the top-level schematic file, without path qualifiers and without extension.

The -*target* parameter specifies the desired type of simulation output file, and is one of the following:

- **-vlfunc** — produces a Viewlogic functional simulation file (*design_name*.vsm) based on your source schematic.
- **-vltime** — produces a Viewlogic timing simulation file (*design_name*.vsm) based on the *design_name*_tim.xnf netlist.
- **-verilog** — produces a structural Verilog HDL file (*design_name*_tim.v), an SDF-formatted timing back-annotation file (*design_name*_tim.sdf) and a stimulus template file (*design_name*_tim.stim), based on the *design_name*_tim.xnf netlist, for use with any Verilog-compatible simulator.

- **-vss** — produces a structural VHDL file (*design_name*_vss.vhd) and an SDF-formatted timing back-annotation file (*design_name*_vss.sdf), based on the *design_name*_tim.xnf netlist, for use with the Synopsys VSS simulator.

For Viewlogic functional simulation, you can use Viewlogic's vsm command on your source schematic to prepare the .vsm file if your design contains no behavioral modules. However, the initial states of registers in your simulation will not reflect any INIT attributes you may have specified in your schematic. If your design contains behavioral modules or you want to simulate non-zero initial register states, you should use the **`xepldsim`** command to prepare your vsm file.

To perform timing simulation using any other simulator tool than those listed above, use the XNF netlist translator provided by the simulator manufacturer and use the *design_name*_tim.xnf file as the input file.

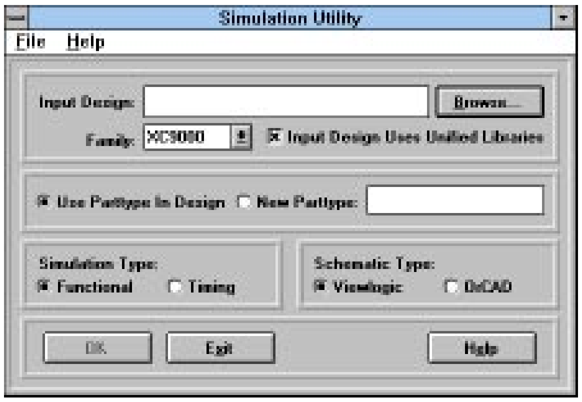
Note: When the fitter processes your design, some of your original nodes may be removed or replaced due to logic optimization. Such nodes cannot be viewed or stimulated during timing simulation. All of the device I/O signals are always maintained.

Simulation on a PC Under Windows

In your Xilinx program group you will find an icon labelled "Simulation Utility." Double-click on this icon to enter the Simulation Utility Control Panel. To prepare a simulation file using the Simulation Utility, follow these steps:

1. From the Xilinx program group, double-click on the Simulation Utility icon. The control panel shown in Figure D-1 will appear.
2. If the correct **`Input Design`** is not listed on panel, click once on the **`Browse`** key and select the correct design.
3. Make sure the **`Family`** is correctly set to XC7000 or XC9000.
4. Select either **`Functional`** or **`Timing`** under **`Simulation Type`**.
5. Select either **`Viewlogic`** or **`OrCAD`** as the **`Schematic Type`** and click on **`OK`**. A simulation report screen will appear and keep you advised of the progress ot the simulation.

For Viewlogic designs the Simulation Utility produces a Viewsim network file, *design_name*.vsm. For OrCAD designs it produces a VST-formatted file for the OrCAD 386+ simulator, *design_name*.vst.



Input Design:
Family: XC9000
[✓ Input Design Uses Unified Libraries]
[✓ Use Parttype In Design] New Parttype:
Simulation Type: [✓ Functional] Timing
Schematic Type: [✓ Viewlogic] OrCAD

Figure D-1. Simulation Utility Control Panel

Simulation on a PC using DOS

The xsimmake command prepares a functional or timing simulation model for Viewlogic or OrCAD 386+. On the DOS command line, enter:

```
xsimmake -f target design_name
```

The `target` parameter is one of the following:

- oef6 — produces an OrCAD functional simulation file (*design_name*.vst).
- oet6 — produces an OrCAD timing simulation file (*design_name*.vst).
- vef6 — produces a Viewlogic functional simulation file (*design_name*.vsm).
- vet6 — produces a Viewlogic timing simulation file (*design_name*.vsm).

Simulating Power-On Initialization

In both the functional and timing simulation files produced by the **`xepldsim`** command, **`xsimmake`** command or **`Simulation Utility`** icon, a global net named PRLD is added to the design to represent the device power-on condition. When the PRLD input is pulsed, all registers in the device are initialized to the states specified by INIT attributes in your design (default is zero for XC9000).

To simulate the device power-on condition, set the PRLD net input to the High state at time zero. Set PRLD Low after any positive time interval. Registers are initialized instantaneously (zero delay from PRLD to registers) and are held at the initial state as long as PRLD is High. Registers are allowed to change state in response to user stimulus any time after PRLD is set Low. Before setting PRLD Low, you should set all essential device inputs to valid logic levels to prevent registers from lapsing back into the “unknown” state.

Numerics

3-state
- vs. multiplexing, 4-2

A

altran, 2-24
Altran command, A-1
Answers to common questions, A-1, D-1
applications, 4-1
attributes, B-1, C-1
- INIT, B-4
- LOC, B-2
- LOWPWR, B-3
- PART, 3-1, B-1
- PLD, B-2

B

behavioral modules, 2-15, 2-16
bidirectional signals, 4-2
bidirectionals, 2-16
boxes in Viewlogic schematics, A-1
BUF, 2-5
BUFE, 2-5
BUFG, 2-8, 2-10, 2-11
BUFTs, 2-5
buses, bidirectional, 4-2

C

clocks
- global, 2-7

common problems, how to solve, A-1, D-1
common symbols, 2-2
component not found message, A-2
components
- common, 2-2
- custom
  - example of, 2-14, 2-20
- non-EPLD, finding, A-1

custom component
- examples, 2-14, 2-20

custom symbols, 2-17

D

design
- applications and techniques, 4-1
- device-independent, 2-1
- example, 1-3
- fitting, 1-11, 1-12
- FPGA to EPLD conversion, 2-20
- hierarchical, 2-19
- procedure, 1-3
- speed optimization, 3-13
- tradeoffs, 3-1

device
- selecting, 3-1, B-1
- selection, 1-12

device selection
- automatic selection criteria, 3-3

device-independent design, 2-1

E

EPLD design
- converting from FPGA, 2-20

errors
- component not found, A-2

example design, 1-3

F

fitter reports, 1-14
fitting
- problems and solutions, 3-1, A-2
- Viewlogic, 1-11, 1-12

FPGA design
- converting to EPLD, 2-20

functional simulation
- Viewlogic, 1-8

G

global clock nets, 2-7, 2-9, 2-11
GND, 2-4
ground signals, 2-3

H

hierarchical design, 2-19

I

IBUF, 2-16
INIT attribute, B-4
internal nodes and timing simulation, A-3, D-6
IOPAD symbol, 2-16

L

library
  schematic, 2-1
  unified, 2-1
LOC attribute, B-2
logic
  optimization, B-3, B-4
  reducing levels of, 3-19
LOWPWR attribute, B-3

M

macro component
  creating custom, 2-19
macrocell feedback, 2-5
messages
  component not found, A-2
multiplexing vs. 3-state signal, 4-2

N

nodes, internal, and timing simulation, A-3

O

OBUF, 2-6, 2-16
OPAD, 2-6
optimization, B-3, B-4
  effects on internal nodes, D-6
  for speed, 3-13

P

package
  selection, 1-12
part
  selection, 1-12
PART attribute, 3-1, B-1
pin assignment, B-2
PinSave file, 3-8
PLD
  attribute, B-2
  linking symbol with with PLUSASM file, B-2
PLUSASM file
  linking with PLD symbol, B-2
power signals, 2-3
power, controlling, B-3
predefined groups, T-Spec, 3-17
primitives
  definition, 2-2
PRLD (preload) signal
  control of, B-4
  registers stuck at preload value, A-2
problems, common, how to solve, A-1, D-1
PROcapture, 1-7
procedure for basic design, 1-3
programming (device)
  Viewlogic, 1-11, 1-12
PROsim, 1-9

Q

Questions commonly asked, A-1, D-1

R

read-back registers, 4-1
registers
  read-back, 4-1
  stuck at preload values, A-2
reports
  from fitter, 1-14

S

schematic
  design, getting started, 1-1
  library, 2-1
simulation
  functional
    Viewlogic, 1-8
  problems and solutions, A-2
  timing
    and internal nodes, A-3

Viewlogic, 1-15, 1-16
speed
setting, 1-12
speed optimization, 3-13
state machines
FPGA to EPLD conversion, 2-26
symbols
common, 2-2
custom, 2-17
GND, 2-4
IOPAD, 2-16
non-EPLD, finding, A-1
unused inputs, 2-3
VCC, 2-4
SymGen command, 2-17

T

target device, selecting, 3-1, B-1
TIMEGRP
symbols, 3-17
TIMESPEC primitive
basic groups, 3-17
FROM-TO statement, B-4
timing simulation
and internal nodes, A-3
Viewlogic, 1-15, 1-16
TNM attribute, 3-17
tradeoffs, in fitting a design, 3-1
TS attribute, 3-17
length, 3-16
time delay units, 3-18
T-Specs, 3-14

U

unified library, 2-1

V

VCC, 2-4
ViewDraw
configuration, 1-5
Viewlogic
functional simulation, 1-8
timing simulation, 1-15, 1-16

W

white boxes, in Viewlogic schematics, A-1

X

XACT–Performance, 3-14
predefined groups, 3-17
TS attribute, 3-16
Xaltran, 2-24
XDraft command, 1-7, 2-24
XOR, registered, 2-14, 2-20
.XSF file, 2-17