CPLD Schematic Design Guide

CPLD Schematic Design Guide — 2.1i *Printed in U.S.A.*

About This Manual

This *CPLD Schematic Design Guide* provides information on using the CPLD fitter 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 CPLD with CAE tool interfaces, refer to the following:

Libraries Guide

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

CPLD Synthesis Design Guide

Additional Resources

For additional information, go to http://support.xilinx.com. Use the search function at the top of the support.xilinx.com page or click links that take you directly to online resources.

The following table provides information on tutorials and some of the resources you can access using the support.xilinx.com advanced Answers Search function.

Resource	Description/URL
Tutorial	Tutorials covering Xilinx design flows, from design entry to verification and debugging http://support.xilinx.com/support/techsup/tutorials/index.htm
Answers Database	Current listing of solution records for the Xilinx software tools Search this database using the search function at http://support.xilinx.com/support/searchtd.htm

Resource	Description/URL
Application Notes	Descriptions of device-specific design techniques and approaches http://support.xilinx.com/apps/appsweb.htm
Data Book	Pages from *The Programmable Logic Data Book*, which describe device-specific information on Xilinx device characteristics, including readback, boundary scan, configuration, length count, and debugging http://support.xilinx.com/partinfo/databook.htm
Xcell Journals	Quarterly journals for Xilinx programmable logic users http://support.xilinx.com/xcell/xcell.htm
Tech Tips	Latest news, design tips, and patch information on the Xilinx design environment http://support.xilinx.com/support/techsup/journals/index.htm

Manual Contents

This manual covers the following topics:

- Chapter 1, “Getting Started with Schematic Design” chapter, presents an overview of schematic design for CPLD devices, including a simple example design.
- Chapter 2, “Design Entry Techniques” chapter, describes the fundamental techniques for expressing logic in a schematic design for a CPLD device.
- Chapter 3, “Controlling Design Implementation” chapter, discusses techniques for controlling how various parts of your design are implemented into a CPLD device.
- Chapter 4, “Design Applications” chapter, describes useful techniques for expressing efficient CPLD designs.
- Appendix A, “Attributes” lists and describes CPLD schematic attributes, which allow access to CPLD architectural features.
- Appendix B, “CPLD Library Selection Guide” lists all the symbols that may be used in CPLD schematic designs.
- Appendix C, “Fitter Command and Option Summary” summarizes implementation options of the Design Manager, and lists parameters for the the **`cpld`** command.

- Appendix D, “Simulation Summary” shows how to sumulate from the Design Manager and command line.

Conventions

This manual uses the following typographical and online document conventions. An example illustrates each typographical convention.

Typographical

The following conventions are used for all documents.

- `Courier font` indicates messages, prompts, and program files that the system displays.

  ```
  speed grade: -100
  ```

- **`Courier bold`** indicates literal commands that you enter in a syntactical statement. However, braces "{ }" in Courier bold are not literal and square brackets "[ ]" in Courier bold are literal only in the case of bus specifications, such as bus [7:0].

  `rpt_del_net=`

  `Courier bold` also indicates commands that you select from a menu.

  `File` → **`Open`**

- *Italic font* denotes the following items.

  - Variables in a syntax statement for which you must supply values

    `edif2ngd` *design_name*

  - References to other manuals

    See the *Development System Reference Guide* for more information.

- Emphasis in text

  If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.

- Square brackets "[ ]" indicate an optional entry or parameter. However, in bus specifications, such as bus [7:0], they are required.

  edif2ngd [*option_name*] *design_name*

- Braces "{ }" enclose a list of items from which you must choose one or more.

  lowpwr ={**on**|**off**}

- A vertical bar " | " separates items in a list of choices.

  lowpwr ={**on**|**off**}

- A vertical ellipsis indicates repetitive material that has been omitted.

```
IOB #1: Name = QOUT'
IOB #2: Name = CLKIN'
.
.
.
```

- A horizontal ellipsis ". . ." indicates that an item can be repeated one or more times.

  allow block *block_name loc1 loc2* . . . *locn;*

Online Document

The following conventions are used for online documents.

- Red-underlined text indicates an interbook link, which is a cross-reference to another book. Click the red-underlined text to open the specified cross-reference.
- Blue-underlined text indicates an intrabook link, which is a cross-reference within a book. Click the blue-underlined text to open the specified cross-reference.

Chapter 1

Getting Started with Schematic Design

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

- “Overview of Schematic Design Methods”
- “Design Flow Summary”
- “Schematic Design Flow Example”

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 the “Design Entry Techniques” chapter. The following figure summarizes the design flow.

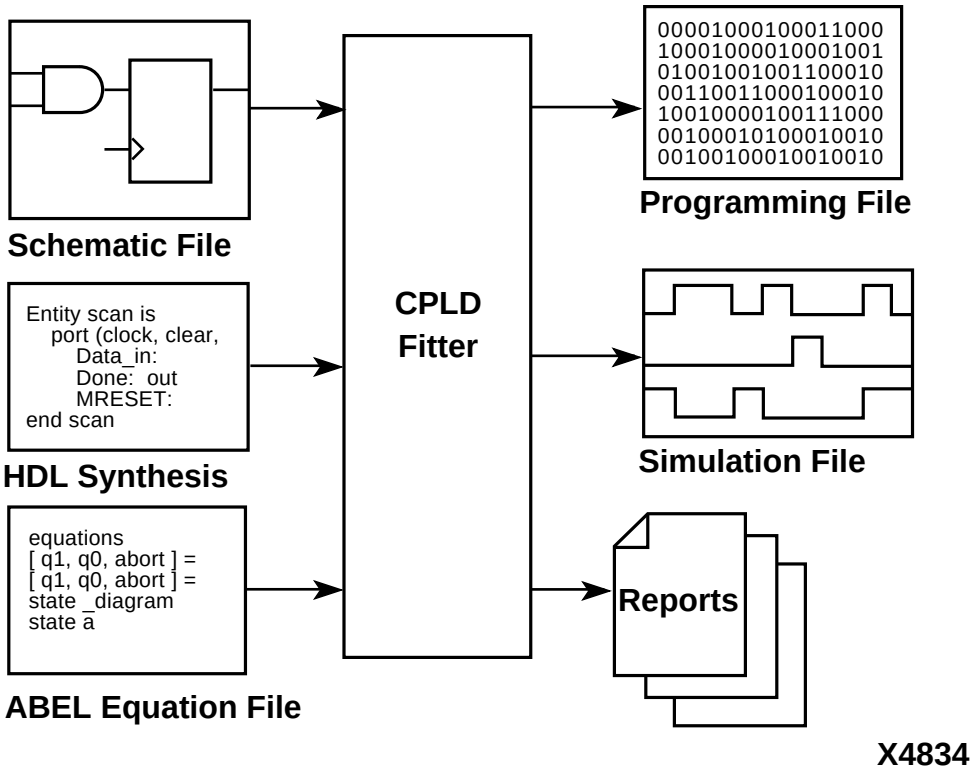

Figure 1-1 Basic Schematic Design Flow

Currently, the Viewlogic, Mentor and Cadence software packages are directly supported by the Xilinx CPLD library and interface for CPLD design entry and simulation. Xilinx also provides the Foundation development system. Other compatible interfaces and CPLD libraries may be available from their manufacturers.

Design Flow Summary

The Design Manager / Flow Engine takes EDIF netlist, XNF or PLD files from your design tool and fits them onto Xilinx devices. You can select a specific device or let the Design Manager select a device for you, based on the most economical solution that will satisfy the functional and timing parameters of the design.

Generated Reports

By default the fitter produces the following significant output files:

- Fitting 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 cpld command for the same design name. (The Guide file is written only upon successful completion of the fitter.)
- Programming file *(design_name*.jed for XC9000) — is a JEDEC-formatted (9k) programming file to be down-loaded into the cpld device.
- Timing simulation database *(design_name*.nga) — a binary database representing the implemented logic of the design, including all delays, consisting of Xilinx simulation model primitives (simprims).

The Design Manager contains a Report Browser for examining selected reports. If you have already run the fitter, the Report Browser contains the Fitting Report and the Translation Report, and, if you have selected timing simulation options, it also contains simulation reports. To access the Report Browser from the Design Manager:

`Utilities → Report Browser`

After the Report Browser displays, to read any of the reports simply double-click the appropriate report icon.

Timing Simulation

The Design Manager optionally produces timing simulation data when you implement your design, and produces either an EDIF, VHDL or Verilog HDL formatted netlist for your simulator..

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:

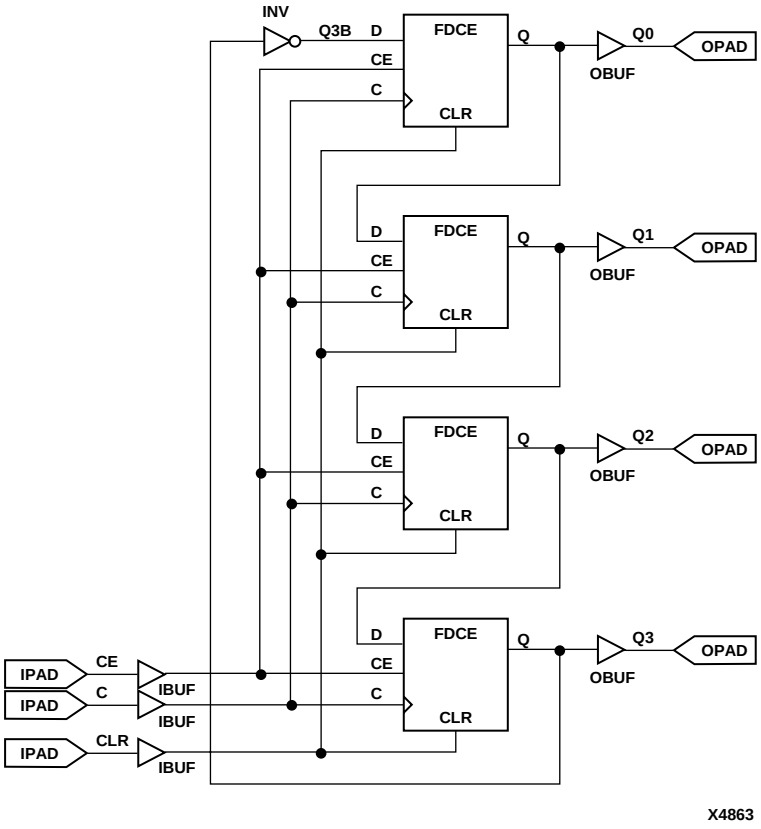

Figure 1-2 Example 4-Bit Johnson Counter Design

Simulation results for this design are shown in the "Example Viewlogic Functional Simulation Results" figure.

The design entry and simulation steps are summarized for Viewlogic and Mentor software. Other supported schematic design software

has similar procedures; refer to the appropriate Xilinx interface guide, if applicable, or the manufacturer's documentation.

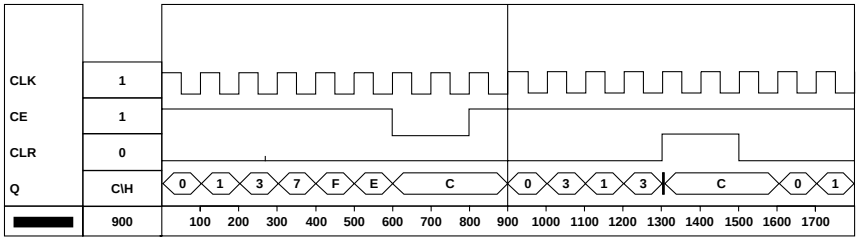

Figure 1-3 Example Viewlogic Functional Simulation Results

Configuring the Design Entry Tool

Many design entry tools have a project management facility that you can use to create a working directory for your design and to select the vendor component libraries to use in your design.

Workview Office Project Manager

1. Call up the Viewlogic Project Manager by selecting the Project Manager icon in the Workview Office icon group. Create a new project named **`jcount`**.

   `File` → **`New`**

2. Select a directory and name the new project **`jcount`**.

3. Call up the libraries you need to create your design.

   `Project` → **`Libraries...`**

4. The Library Search Order dialog box displays. Use this tool to add the XC9000 library, plus the **`builtin`** and **`simprims`** libraries.

5. Use the **`Browse`** key to select directories and the **`Add`** key to add libraries. For example, browse to *installation_path*/**`viewlog/data`**/**`xc9000`** (for **`XC9500`** target devices) where *installation_path* is the root directory where the Xilinx software package was installed. Then click **`Add`** and the xc9000 libraries

will be added to the list. When you have all the libraries you need for the project, click **OK**.

Viewlogic On Workstations

1. To create a working directory for your design in Viewlogic's Powerview, use the **Project** → **Create** command.
2. Configure the design entry tool to access the Xilinx CPLD component library for schematics you develop in the project you just created. In Powerview, 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* / **viewlog/data** / **xc9000** (for XC9500 target devices), where *installation_path* is the root directory where the Xilinx software package was installed. Under the "Library" column, enter **XC9000**, which is also known as the library alias. Under the Type column, select **Megafile** (compressed read-only format).
3. 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 [m] *installation_path*/**viewlog/data/xc9000** (xc9000)

   where *installation_path* is the root directory where the Xilinx software package was installed.
4. If you plan to simulate using Viewsim, you also need to include the Xilinx “simprims” and Viewlogic "builtin" library in the Search Order window or the viewdraw.ini file.

   DIR [m] *installation_path*/**viewlog/data/simprims** (simprims)

   DIR [m] *installation_path*/**viewlog/data/builtin** (builtin)

Mentor

1. Call up the Mentor Design Manager as follows:

   ```
   pld_dmgr
   ```
2. Select the **Tools** icon. Then go to **Tools** program group and select **pld_da** (Design Architect).

Drawing the Design

1. Invoke the schematic drawing tool and draw the design.
   - If you are using Workview Office or Powerview, invoke the **`ViewDraw`** tool.
   - If you are using Mentor Graphics you would invoke **`pld_da`** (Design Architect).
2. If you prefer to use the completed schematic shown in "Example 4-bit Johnson Counter" figure as a sample design, copy the jcount schematic file from the examples directory of the Xilinx software.
   - For Viewlogic, copy all files and subdirectories under the *installation_path*\viewlog\tutorial\jcount directory into your local directory (the schematic file is jcount.1 under the sch subdirectory).
   - For Mentor select the **`Find Comp`** icon in the Design Architect and browse to *installation_path*\mentor\tutorial\jcount, then select jcount.

   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 XC9000 library. For example, 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 the "Design Entry Techniques" chapter.

   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 appear in the fitter reports as your pin names and are used as your top-level signal names during timing simulation, after design implementation.
3. Save your schematic when finished. The Viewdraw **`Write`** command performs schematic rule checking and writes a "wire-list" file in the wir directory (wir/jcount.1).

Performing Functional Simulation (Optional)

Xilinx schematic capture libraries contain simulation models allowing you to perform functional simulation directly from your

schematic. In most libraries, models for all registered symbols contain a global signal named PRLD which, when pulsed high, initializes all the flip-flops inside the symbol's model. Remember to pulse the PRLD signal high and drive all your top-level input signals (pins) to valid logic levels before running your simulation vectors.

Viewlogic

If you are using Viewlogic, Xilinx provides a Viewsim command file for the jcount design that can be found in *installation_path*/viewlog/tutorial/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.

Mentor

You can functionally simulate XNF or EDIF based designs by using **`pld_xnf2sim`** or **`pld_edif2sim`** to convert the designs to a Mentor simulation model. The EDIF design must be Xilinx compatible and expressed in Unified Library components.

Performing functional simulation on a pure schematic design consists of creating a viewpoint in **`pld_dve`** from the schematic that you

created in Design Architect and using **pld_quicksim** to simulate the design. For more information see the *Mentor Graphics Interface Guide*.

Implementing the Design

Before implementing the design, your schematic must first be translated into an EDIF 2.0.0 formatted netlist.When netlisting, make sure the netlist hierarchy stops at the Xilinx library primitives and does not expand into any functional simulation models that may exist beneath the Xilinx primitive symbols.

Creating an EDIF Netlist

From *Viewlogic Workview Office:*

1. Go to the ViewDraw tool and select:

   Tools → **Export EDIF**

2. The EDIF Interfaces Dialog Box appears. Select the **EDIF Netlist Writer** tab and **Browse** to the **jcount.1** schematic. Enter **xilinx** in the level field.

3. To create the **.edn** file click once on **Apply**.

For *Viewlogic Powerview:*

1. Invoke the **edifneto** command

2. Specify "XILINX" as the "level" attribute ("**-l xilinx**" option on the **edifneto** command line).

From *Mentor* you must convert to EDIF with the **pld_men2edif** utility before implementing the design with the Design Manager.

To convert your design to EDIF, follow these steps.

1. In the Mentor Design Manager, double-click the left mouse button on the pld_men2edif icon.

   The dialog box labeled "Mentor to EDIF Netlist appears.

2. In the Component Name field, enter the component name, or click **Navigator** to browse a list of design names.

3. In the From Viewpoint field, you can enter the viewpoint name if you do not want to use the default viewpoint. Alternatively, in step 2 you can select a viewpoint under the component.

4. Select the appropriate architecture for your design in the PLD Technology field.
5. Select the appropriate Bus Dimension Separator Style.

   This is important if you are merging components from other design-entry tools into a single design. Choosing a bus-index delimiter lets you insure that the bus-index delimiters that pld_men2edif writes out are consistent with those of any other design-entry tools with which you are interfacing. Mentor EDIF uses parentheses. Synopsys EDIF uses angle brackets.
6. Click **OK**.

   Pld_men2edif now produces an EDIF file that you can submit to the Xilinx Design Manager. The output name is *component_name*.edif.

Implementing from Design Manager

1. Start the Xilinx Design Manager. From the Windows Program Manager click the Design Manager icon. If you are using the UNIX command line enter the following command.

   dsgnmgr
2. From the Design Manager, create a new project and select **jcount.edn** as its input.

   File → **New Project**
3. The **New Project** dialog box appears. Select **Browse** for **Input Design** and find **jcount.edn**, then click **OK**. The project should appear in the Design Manager.
4. Implement the design.

   Design → **Implement**

   The **Implement** dialog box appears. This box allows you to select a specific Xilinx device to implement your design.
5. Click once on **Select** and select for XC9500, XC9500XL or XC9500XV Family; leave Device, Package and Speed Grade set to **AUTO**.
6. Click **OK** when you have finished.
7. Click **Run** to implement the design.

Examining 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 fitter. If you are using a Design Manager 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:

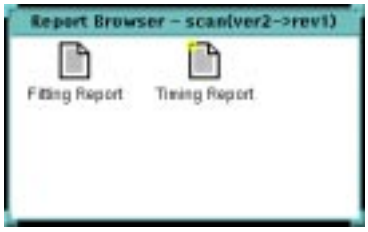

Figure 1-4 Report Browser

- 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.

Performing Timing Simulation

To perform timing simulation you must extract a new EDIF netlist from the implemented design. To avoid overwriting, you may want to specify and output filename different than your design entry netlist.

The Design Manager optionally produces timing simulation data when you implement your design.

1. To produce timing data go to the options menu:

   `Design` → **`Implement`**

2. The **`Implementation`** menu will appear. Click once on the **`Options`** key to get the **`Options`** dialog box.

3. Select **`Produce Timing Simulation Data.`**
4. Go to the **`Interface`** template and select **`EDIF`** as the simulation netlist format. When the fitter runs it generates the appropriate data. If you have already run the fitter, go back to the **`Flow Engine`**; a **`Timing`** block will appear in the flow.
5. Press the forward arrow to resume the program from the **`Fit`** block.

Workview Office and Powerview

In most cases, you can use the same command file you used during functional simulation to perform the timing simulation. You may need to make minor adjustments to the command file used to functionally simulate the design before it can be used to perform a timing simulation.

The typical procedure for performing a timing simulation is as follows.

1. Import the EDIF file with timing information produced by the fitter to create a wire-list file.
2. Create the timing simulation network (VSM file).
3. Start ViewSim.
4. Load the VSM file into ViewSim.
5. Simulate the device's startup sequence.
6. Execute the command file used during functional simulation. For the JCOUNT sample design, a Viewsim command file can be found in *installation_path*/viewlog/tutorial/jcount/jcount.cmd
7. VewTrace is automatically opened in response to the WAVE command. View the waveforms produced by the simulation.
8. Repeat steps 5 and 6 as necessary to verify the timing information.

See the *Viewlogic Interface Guide* for detailed information on each of the above steps.

Mentor

During design implementation, the Xilinx Design Manager can produce an EDIF (EDN) file. For EDIF files, the process of timing simulation consists of converting the EDIF netlist to a Mentor EDDM model, creating a component and a viewpoint, and simulating the design with **pld_quicksim**.

1. Double-click the left mouse button on the **pld_edif2tim** icon in the Mentor Design Manager Tools window.

   The dialog box labeled EDIF to Mentor Eddm Sing Object appears.

2. Enter the name of the EDN file in the EDIF Input File field, or click **Navigator** to browse the available files.

   The component created from the EDN file is put into a design library called *my_design_lib*. If you have already implemented your design at least once, this directory already exists. If you don't wish to copy over it, move it to another directory before proceding.

3. Click **OK**.

4. Invoke DVE, by double-clicking the left mouse button on the **pld_dve** icon in the Mentor Design Manager Tools window.

   The Pld_dve Dialog Box appears.

5. Enter the top-level component name created by **pld_edif2tim** in the *my_design_lib* directory.

6. Use the Navigate button to navigate all the way down to the "default" viewpoint and select the viewpoint.

7. Select the Simulation Button.

8. Select the appropriate technology from the PLD Technology box.

9. Click **OK**.

You can now submit the design to **pld_quicksim** for timing simulation.

1. To invoke **pld_quicksim**, double-click the left mouse button on the **pld_quicksim** icon in the Design Manger Tools window.

The **pld_quicksim** dialog box appears. For more detailed information on the dialog box options, refer to the Mentor Graphics documentation.

2. In the Design field, enter the name of the top-level design created by **pld_edif2tim**.
3. In the Select Desired Mode field, select **Cross-Probing**.

   Normally, you select cross-probing for back-end EDDM designs but not for front-end designs. You can only cross-probe back-end designs that contain either pure schematic or schematics that contain EDDM hierarchical models. You cannot cross-probe designs written in HDL or that contain HDL models.

Warning: In order for cross-probing to work, other sessions of Design Viewpoint Editor and QuickSim must be closed. Otherwise, the interprocess communication gets confused. This includes another user's session invoked by rlogin from another workstation.

4. Set the timing modes as desired.
5. Click **OK**.

   Pld_quicksim now simulates the design. The QuickSim graphical user interface appears. If you selected cross-probing, DVE is invoked as well.

6. In DVE, open the viewpoint of the front-end schematic design, that is, the viewpoint submitted to **pld_men2edif**.
7. Open the sheet of the design, and select signals that you wish to trace.

   These signals are automatically added to the QuickSim trace window. If you have a file that sets up your trace window and stimulus, use that instead. Any signals selected in the trace window will select the same on the schematic on which they reside in the DVE window. If such sheets have not been opened in DVE yet, you must open them to see the selections.

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 *JTAG Programmer Guide*.

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. This chapter includes the following sections:

- "Library Symbols"
- "Input / Output Buffers"
- "LogiBlox Modules"
- "Behavioral Modules"
- "Hierarchical Design"
- "Retargeting a Design From a Different Family"

Library Symbols

The Xilinx XC9000 library contains all component symbols used by the Xilinx XC9500, XC9500XL, and XC9500XV device families. While most symbols of the library are common to all families, there are some symbols which are specific to CPLDs.

Physically, each major Xilinx device family (for example, 3000, 4000, 9000) has its own schematic library, implemented for each of the supported schematic entry tools. For each tool, there is a library directory for the XC9000 device family, which supports XC9500, XC9500XL, and XC9500XV devices. 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 a CPLD device, or any subsheet 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.

For example, 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
- LogiBLOX 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 pre-defined functionality, but often their implementation is subject to the optimizations performed by the fitter software. Macros are always expanded into the netlist during schematic-to-netlist translation before the netlist is read by the fitter. EDIF netlists may either be in hierarchical or flattened form; this has no impact on the fitter's performance.

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 previously.

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 CPLD 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 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.

LogiBLOX modules are high-level, customizable library macros that are available for some schematic entry tools. LogiBLOX modules are described 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, or asynchronous clear (CLR) or preset (PRE) inputs on flip-flops and other registered macros. Unconnected AND-gate, OR-gate, CLR and PRE inputs are simply discarded, as if the component 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. Timing simulation after fitting 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.

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 tristate 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 the “Bidirectional I/O Pin” figure.

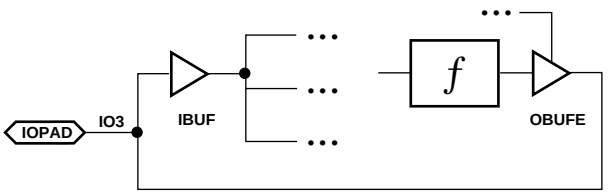

Figure 2-1 Bidirectional I/O Pin

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 appear in the fitter reports as your pin names and are used as signal names during simulation.

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

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) of the CPLD whenever possible.

A clock input signal may pass through an inverter to perform negative-edge clocking and the fitter can still use a global clock pin to implement it. The same global 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 “Input CLK1 can be Optimized onto a Global Clock Pin (GCK)” figure. Global clock inputs may also be used as ordinary input signals to other logic elsewhere in the design.

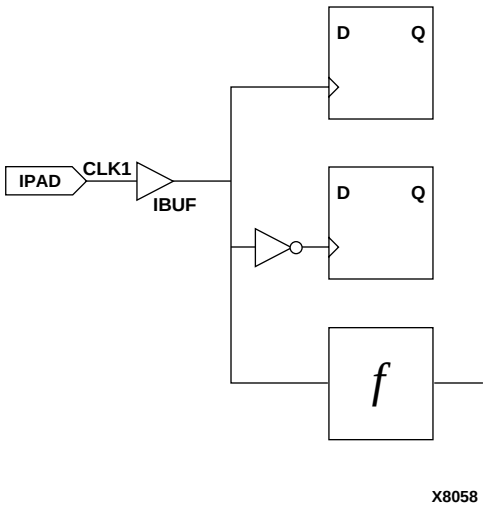

Figure 2-2 Input CLK1 can be Optimized onto a Global Clock Pin (GCK)

If a device input passes through any logic function (other than an inverter) before it is used as a clock by any flip-flop, the input cannot be routed using the global clock path. Instead, the clock signal will be routed through the logic array for all flip-flops clocked by such a device input.

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. You could alternatively apply the BUFG=CLK attribute to an IBUF symbol or the pad net.

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 mapping IBUF symbols to the global clock pins.

1. On the Design Manager graphical interface, go to the Flow Engine and select **`Setup`** → **`Options`**
2. The Design Implementation Option menu appears. Select **`Edit Template`**

3. Then select **Basic Options**
4. Place a check on the **Off** box adjacent to **Use Global Clocks**.

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

Output Enable Signals

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

A global tristate control input signal may pass through an inverter or control the disable input (T) of an OBUFT symbol to perform an active-low output-enable. The same tristate control input may even be used both inverted and non-inverted to enable alternate groups of device outputs, as shown in the “Input OE2 can be Optimized onto a Global Tristate Control Pin (GTS)” figure. Global tristate control inputs may also be used as ordinary input signals to other logic elsewhere in the design.

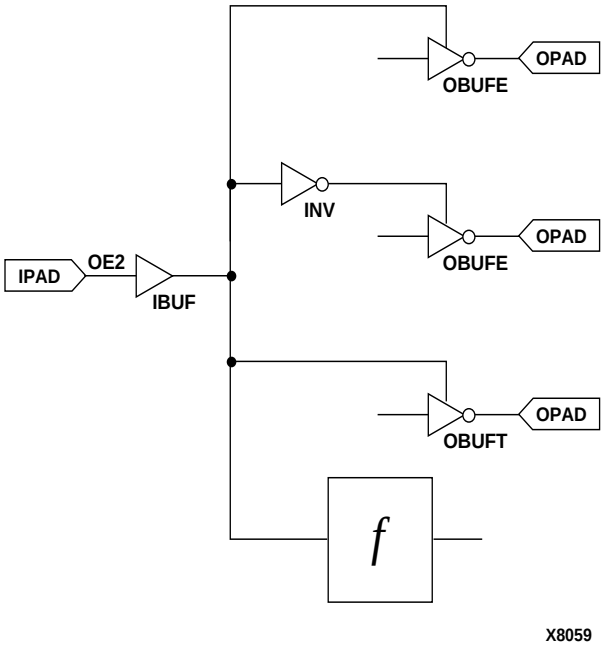

Figure 2-3 Input OE2 can be Optimized onto a Global Tristate Control Pin (GTS)

If a device input passes through any logic function (other than an inverter) before it is used as a tristate control by any output, the input cannot be routed using the global tristate control path. Instead, the output enable signal is routed through the logic array, for all device outputs controlled by such an input.

There are a limited number of global tristate control pins on each XC9500 device (consult the device data sheet). If you need to explicitly control the use of global tristate control pins, you can use the BUFGTS symbol. You can alternatively apply the attribute BUFG=OE to an IBUF symbol or the pad net.

The global tristate control pins provide much shorter input-to-output-enable delays than tristate controls routed through the logic array. Routing a tristate 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 tristate control pins.

1. In the Design Manager, go to the Flow Engine and select **Setup** → **Options**
2. The Design Implementation Option menu appears. Select **Edit Template**
3. Then select **Basic Options**
4. Disable the box adjacent to **Use Global Output Enables**.

If global output enable optimization is disabled, IBUF inputs used for tristate control always pass through the logic array. You can still use BUFGTS symbols or the BUFG=OE attribute to explicitly specify global tristate 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. 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.

If a device input passes through any logic function other than an inverter before it is used as an asynchronous clear or preset by any flip-flop, the input cannot be routed using the global set/reset path. Instead, the clear or preset signal will be routed through the logic array for all flip-flops controlled by such an input. 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 XC9500 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. You can alternatively apply the attribute BUFG=SR to an IBUF symbol or the pad net.

You can prevent the fitter from using the global set/reset pin. In the Design Manager, disable the **Use Global Set/Reset** option in the **Basic Options** template of the Flow Engine.

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.

Clock Enable

When targeting an XC9500 device, any FDCE or FDPE primitives in your design will be expanded into an ordinary D-type flip-flop with its Q-feedback multiplexed into its D-input. This implementation will be similar to the way the FDCPE macro is expanded in the XC9000 schematic library.

When targeting an XC9500XL device, logic connected to the clock enable (CE) input of FDCE and FDPE primitives in your design will be unconditionally implemented using the clock enable p-term of the XC9500XL macrocell. Only the FDCE and FDPE primitives use the clock enable p-term.

If you use FDCE or FDPE components and target an XC9500XL device, you may find that the logic connected to the clock enable input on some components may not get optimized into the same macrocell as the flip-flop. The XC9500XL macrocell contains only a single product-term to implement clock enable input logic. The CPLD fitter does not attempt to transform your clock enable input logic onto the D-input of the flip-flop if it cannot be completely implemented using the clock enable p-term. In general, only primary inputs (device input pins or macrocell feedbacks), their complements or positive-logic AND-gate functions of primary inputs or their complements can be completely implemented using the p-term. If you connect a more complex logic function to the clock enable input of an FDCE or FDPE component and it does not get completely implemented on the clock enable p-term, your design may incur extra macrocell resources and combinatorial macrocell feedback delays.

If you have an existing CPLD schematic containing FDCE or FDPE components and you do not want the logic connected to the CE input

of the components to be implemented using the clock enable p-term in the XC9500XL macrocell, you can simply replace FDCE or FDPE components in your schematic with FDCPE components from the XC9000 library. The FDCPE component is a macro which always gets expanded into a simple D-type flip-flop with its Q-feedback multiplexed into its D-input; the clock enable p-term is not used. After substitution, the unconnected PRE or CLR input to the FDCPE is automatically trimmed away by the CPLD fitter.

Tristate Multiplexing

XC9500 CPLD devices can emulate tristate 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 tristate signals can be wire-ANDed in the FastCONNECT to emulate bussing and tristate multiplexing. When an on-chip tristate 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 tristate multiplexing (bussing) in the schematic, tie together the outputs of multiple tristate buffer symbols, like BUFE and BUFT, as in the “Correct On-Chip Tristate Multiplexing” figure. 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.

Note: XC9500XL does not support internal tristate bussing. Never use BUFE or BUFT components in an XC9500XL design. On-chip tristate bussing is supported by some of the FPGA device families.

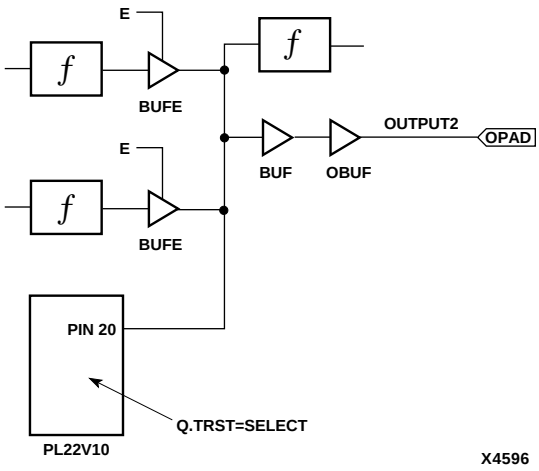

Figure 2-4 Correct On-Chip Tristate Multiplexing

If your design calls for tristate bussing of multiple signals driven off-chip, it may be better to output each signal source on its own tristate output pin and tie the pins together off-chip, as shown in the “Correct Off-Chip Tristate Multiplexing of CPLD Outputs” figure. You cannot connect more than one signal source to the same OBUF or OPAD, as shown in the “Incorrect Tristate Multiplexing of CPLD Outputs” figure.

Note: XC9500XL and XC9500XV devices do not support internal tristate bussing. Never use BUFE or BUFT components in an XC9500XL or XC9500XV design. On-chip tristate bussing is supported by some of the FPGA device families.

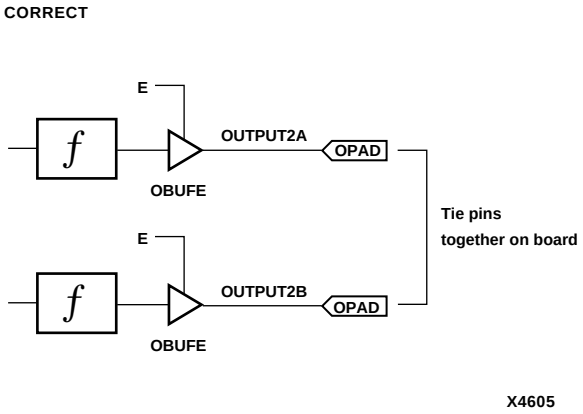

Figure 2-5 Correct Off-Chip Tristate Multiplexing of CPLD Outputs

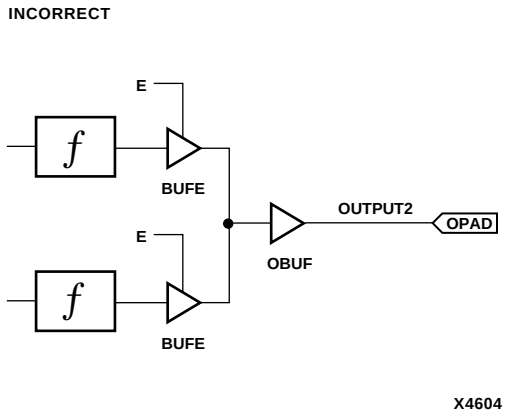

Figure 2-6 Incorrect Tristate Multiplexing of CPLD Outputs

LogiBlox Modules

LogiBLOX is a graphical interactive tool for creating high-level modules, such as counters, shift registers, and multiplexers. LogiBLOX includes both a library of generic modules and a set of tools for customizing them. For detailed information see the *LogiBLOX User Guide*.

The modules you create with LogiBLOX can be used in designs generated with schematic editors from Aldec, Viewlogic, Mentor Graphics, and Cadence.

Use LogiBLOX modules whenever you need a customized version of a standard function. In contrast, a standard ready-made counter has a previously defined set of functions. If you want to use a counter with specific capabilities, you need to have available a library of different counters, one of which contains the functions you need. However, with a LogiBLOX counter, you start with a generic template and tailor its functionality to your needs.

You can use LogiBLOX to design your modules. The Module Selector is a graphical user interface. Use it to tailor modules to your requirements. This is the most common way to design modules in LogiBLOX.

Program Inputs and Outputs

The Module Selector is the LogiBLOX graphical user interface that you use to create a LogiBLOX module. Specifying a LogiBLOX module consists of a) selecting or deselecting optional pins on the symbol, and b) specifying various module attributes. The result is a module customized for a specific function.

After you complete the module specification, LogiBLOX uses its symbol generator, model generator, and netlist generator to create the following three outputs and store them in the current project directory:

- A schematic symbol for inclusion on the schematic

  The symbol generator creates a symbol definition file that your third-party interface converts into a schematic symbol.

- A behavioral VHDL simulation model

The model generator creates a behavioral VHDL simulation model for the LogiBLOX module.

The behavioral model permits the design to be simulated immediately in those environments that support mixed schematic and behavioral simulation.

- Verilog and EDIF gate-level netlists, produced as an alternative simulation medium

  The netlist generator creates a gate-level netlist for the LogiBLOX module that is converted to the third-party's simulation format. These netlists permit immediate simulation of the design in gate-level simulation environments.

Schematic Design Flow

To use the program in a schematic-based environment, follow these steps:

1. Invoke the Module Selector from within your design entry tool.
2. Configure your project directory using the LogiBLOX Setup window.
3. Select a base module type (for example, Counter, Memory, or Shift-register)
4. Customize the module by selecting pins and specifying attributes.
5. After completely specifying a module, click on the OK button. Selecting OK initiates the generation of a schematic symbol and a simulation model for the selected module.
6. Place the module on your schematic.
7. Connect the LogiBLOX module to the other components on your schematic using ordinary nets, buses, or both.
8. Functionally simulate your design at any time.
9. Implement your design with the Xilinx implementation tools.
10. To simulate your design post-layout, convert your design to a timing netlist and use the back-annotation flow appropriate to your CAE tools.

Changing a Schematic Module

To change a module that you have already placed on your schematic, select the module and invoke the Module Selector. The Module Selector displays the settings of the module that you want to edit.

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:

- You may want to re-use existing design solutions that are expressed in Boolean equation or HDL form.
- It is often easier to express combinatorial logic functions and state machines using Boolean equations or HDL than using schematics.

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 EDIF or XNF formatted netlists.

Note: Previous versions of the CPLD fitter used the PLUSASM equation language as an interchange language in place of XNF or EDIF for

some behavioral compilers, such as XABEL. PLUSASM is still recognized and processed by this version of the fitter for the sake of existing behavioral modules based on that interface. However, future versions of the software will not support PLUSASM and it should not be used when creating any new behavioral modules.

If the behavioral compiler tool supports the development of completely behavioral designs, make sure you select the appropriate mode of operation or compilation flow for producing logic modules, not stand-alone designs. The netlist 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.

Note: If you are using a synthesis tool to prepare a behavioral module, make sure you target the appropriate 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 port) 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, either with or 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.

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'
IO pin;
I1 pin;
CLK pin;
Q pin istype `reg';
EQUATIONS
Q := IO $ I1;
Q.C = CLK;
end
```

2. Compile the file to create an EDIF netlist file named regxor.edn.

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

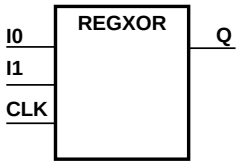

Figure 2-7 The REGXOR Symbol

4. Make sure the symbol is defined as a primitive. For example, if you are using WorkView Office, 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 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**.

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:

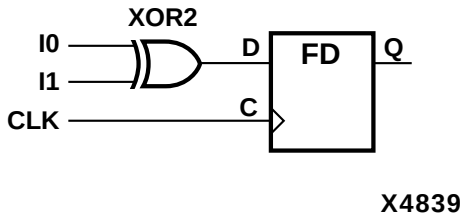

Figure 2-8 The REGXOR Schematic

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

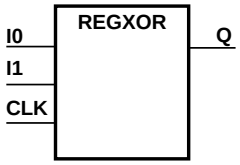

Figure 2-9 The REGXOR Symbol

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 using a CPLD technology library (if applicable).

Schematic Conversion Procedure

When you retarget 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 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 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 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, or other elements that do not have CPLD equivalents, you cannot retarget your design unless you redesign those portions.

Using Workview Office

In this example, we are converting a schematic originally implemented using the XC3000 library into a schematic targeting the XC9000 library.

1. Make sure the Project Manager contains the XC9000 libraries. If they do not use **`Project`** → **`Libraries`** to add them. For example, browse to *installation_path*/**`viewlog/data`**/**`xc9000`** where *installation_path* is the root directory where the Xilinx software package was installed. Then click on **`Add`** and the XC9000 libraries will be added to the list.
2. Close all Workview Office tools except the Toolbar.
3. Open an MS-DOS session and change the current directory to the project directory.
4. Run the ALTRAN command with the following syntax:

   `altran -l` *library old_alias*=*new_alias*

   For example, to change a project from the XC3000 family to the XC9500:

   ```
   altran -l primary xc3000=xc9000
   ```

   Altran will change the library aliases in all of the schematic sheets within the specified library directory. Altran will also modify the alias of the targeted library within the Viewdraw.ini file.
5. After running ALTRAN, open the Workview Office Project Manager. Note that the last library has the XC9000 alias but still

has the path to the XC3000 library. Select **Project→Libraries** to modify the Library Search Order.

6. Select the modified library. In the Path field, change the path so it points to the XC9000 library directory. Click **Change**.
7. Select this library again. Click the **Move Up** button until it is below the primary library and any user-created libraries. Click **OK**.
8. Save the changes in the Project Manager.

Note: When writing the EDIF file from ViewDraw for subsequent implementation in an existing Xilinx project, you must keep the same *design*.**edn** filename and project directory.

In the Design Manager window, select **Design → New Version**. This brings up the New Version dialog box, shown in the "New Version Dialog Box" figure. The version name increments by default.

Change the name of this version, if desired, and add any comments for this version. Click **OK** to create this new version.

Using Viewlogic Powerview on Workstations

In this example, we are converting a Viewlogic schematic originally implemented using the XC3000 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 (XC3000) 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 (XC3000) in the Search Order. For example, if you are converting an XC3000 design to XC9000, your viewdraw.ini file would contain the following two lines:

   DIR [r] *installation_path*/**viewlog/data/xc9000**
   (xc9000)

DIR [r] *installation_path*/**viewlog/data/xc3000 (xc3000)**

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 (XC3000) 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 xc3000=xc9000

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, SLOW, KEEP, BUFG, FILE, NOREDUCE and timing specifications. Change the values of PART, LOC, and PROHIBIT attributes as needed, or remove them.

2. In the Design Manager, create a new Xilinx project for the converted schematic design.

   a) From the Design Manager click the **File** menu and select **New Project**.

   b) Enter a new project name to use for XC9000 implementation.

   c) From the **Target Family** select **XC9500**.

   d) Before processing the design, open the Implementation Options menus and select the options available 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 may 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 CPLD fitter optimizes the logic differently than FPGAs, which makes many of the internal nodes in the design inaccessible. 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, PERIOD and OFFSET.
- FAST and SLOW (output slew-rate control)
- FILE=filename for behavioral modules
- KEEP and COLLAPSE

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

Any attributes contained in the converted design which are not supported by the target CPLD family should be removed from the schematic before netlisting.

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 generally 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 from one-hot encoding may be unnecessary for very simple state machines.
2. If you are using a synthesis tool, recompile the behavioral module specifying XC9000 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 options you select in the Design Manager.

This chapter contains the following sections:

- "Target Device Selection"
- "Controlling Preload Values"
- "Controlling Power Consumption"
- "Controlling Output Slew Rate"
- "Controlling the Pinout"
- "Controlling Logic Optimization"
- "Controlling Timing Paths"

Target Device Selection

By default, the fitter automatically selects a Xilinx device for you, choosing, in general, the smallest part that satisfies the needs and constraints of your design.

1. After you have opened a design, select **`New Device`** from the Implementation menu; the device selection menu appears, as shown in the "Device Selection Menu" figure.

Figure 3-1 Device Selection Menu

2. From this menu, select **XC9500, XC9500XL,** or **XC9500XV** 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 the selected family.

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:

- First, the software selects the smallest die.
- Second, the software selects the smallest package.
- Third, the software selects the fastest speed-grade device for the selected device.

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.

You can also specify a part number in your schematic. Place a CONFIG symbol in your schematic and apply the following attribute to it:

part=*dddd-ss-pppp*

When you open the Part Selector menu in the Design Manager, the Family, Device Package and Speed will all be set to the values you specified in your schematic. You can override these values using the Part Selector menu. The value of the PART attribute in a schematic must be a complete specific part value; you cannot use wildcards.

Controlling Preload Values

The preload values used in the implementation of your design depend on the INIT=R/S attribute. 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). This attribute can be applied to flip-flops or any component containing a register. The initial state of all flip-flops in an XC9000 design is zero unless otherwise specified by INIT=S.

The only differences expected between functional and timing simulation involve the initial states of registers and latches. When you perform functional simulation directly from your schematic, the simulation models for each registered symbol have no knowledge of any INIT attributes you may have attached to the symbol. Therefore, functional simulation assumes that all preload values are zero. Timing simulation uses the actual preload values implemented by the fitter.

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 PWR_MODE 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 or its output net in your schematic:

PWR_MODE=LOW

To specify standard power mode for a function (in case the global power was changed to low), apply the following attribute to a component or its output net:

PWR_MODE=STD

The PWR_MODE attribute can be applied to any logic or flip-flop component in the schematic, including a macro component that has

multiple output signals. The PWR_MODE 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 PWR_MODE attribute is not carried forward by the software. You may therefore need to apply the PWR_MODE 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 PWR_MODE attribute has no effect on components that are not implemented using macrocell logic, such as I/O buffers.

Changing Global Power Mode

To set all macrocells to the Low Power Mode throughout the design, set **Macrocell Power Setting** to **Low** in the **Basic** menu of the **Implementation Options Template** in the Design Manager.

By setting the Power Mode to **Low** in the template, macrocells will operate in low power mode except where you specify the PWR_MODE=STD attributes in the design.

If you want the fitter to automatically select the power mode for individual macrocells based on timing constraints you enter for your design, set **Macrocell Power Setting** to **Timing Driven**. Macrocells involved in timing-constrained paths will have their power settings automatically switched to **Low** only if the low-power propagation delay still allows the macrocell to satisfy all applicable timing constraints. Macrocells that do not participate in timing-constraint paths will operate in standard power mode (**Std**). Applying the PWR_MODE attribute always overrides the **Macrocell Power Setting**.

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 outputs in a CPLD design have fast (unlimited) slew rate. You can change fitter options so that all outputs operate with slow slew-rate or so that the fitter automatically selects slew-rate based on timing constraints you enter for your design. To change output slew rate control for all outputs from the Design Manager, go to the Basic tab of the Implementation Options.

1. From the Design Manager, select **Design** → **Implement**.
2. Click **Options** in the Implement dialog box.
3. Select **Edit Template**
4. Select the tab labelled **Basic**.
5. You can instruct the fitter to automatically switch outputs to slow slew rate by changing the **Output Slew Rate** option to **Slow**.

If you are in the Flow Engine, this procedure is as follows:

1. From the Flow Engine, select **Setup** → **Options**.
2. Select **Edit Template**.
3. Select the tab labelled **Basic**.
4. You can instruct the fitter to automatically switch outputs to slow slew rate by changing the **Output Slew Rate** option to **Slow**.

By setting **Output Slew Rate** to **Timing Driven**, outputs involved in timing-constrained paths will have their slew-rate automatically set to **Slow** only if the slew-rate-limited output propagation delay still allows the output to satisfy all applicable timing constraints. Outputs that do not participate in timing-constrained paths will operated in **Fast** slew-rate mode.

If you want to explicitly force an output to use slow slew rate, apply the SLOW attribute to an OPAD or IOPAD symbol or the pad net.

SLOW

If you want a device output to use fast slew, use the FAST attribute. Simply apply on an OPAD (or IOPAD) symbol or the pad net.

FAST

Applying the FAST or SLOW attribute to an output pad always overrides the **Output Slew Rate** options in the fitter template.

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 fitter to use the committed pinout saved in the guide file.

Xilinx strongly recommends 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 Locking

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

1. In the Design Manager, select an existing design revision that was successfully run through the Fit step (typically, your most recent revision).
2. Select **Design** → **Lock Pins**. The pinout saved in the selected revision (stored in *design_name***.gyd**) is translated into pin location (LOC) constraints and written into a user constraint file (*design_name***.ucf**).
3. Select **View Lock Pins Report** in the dialog box to make sure no pin assignment conflicts were found.
4. When ready, run the fitter (**Design** → **Implement**). The previous pinout information will be read from the UCF file and used in the new design revision.

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.

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 automatically selects 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 lock your pinout yet. Instead, allow the software to redefine the pinout of your modified design. This will continue to give you the greatest flexibility to alter the logic in your design again after you commit your pinout.

Pin Assignment

There are two ways to assign pins. You can use the location box in the Ports Tab window of the Constraints Editor, or you can use the LOC attribute.

Constraints Editor

The Ports Tab window of the Constraints Editor contains a Location dialog box which will create a constraint which locks a user-defined port to a specific device pin.

1. Open the Constraints Editor and go to the Ports Tab Window. To do this, simply click the button labelled “Ports.”
2. In the Location column and in the row associated with the Port Name, double-click the left mouse button. This opens the Location dialog box.
3. In the Location text box, enter a pin identification name.
4. Click **OK**.

See the *Constraints Editor Guide* for more information.

LOC Attribute

You can also assign explicit locations for pins in your design using the LOC attribute in your schematic. To assign a pin location, apply the following attribute to a pad symbol (IPAD, OPAD or IOPAD) or pad net 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 using the Design Manager or the schematic PART attribute (see Target Device Selection in this Chapter). LOC attributes are not always 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.

Ignoring the LOC Attribute

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.

1. Go to the Flow Engine and select **Setup** → **Options**
2. The Design Implementation Option menu appears. Select **Edit Template**
3. Then select the tab **Basic**

4. Remove the check in the box adjacent to **Use Design Location Constraints**. Then click **OK**.

Function Block and Macrocell Assignment

You can explicitly assign internal nodes in your design to specific function blocks or even specific macrocells of the target device. To assign an internal node to a specific location, apply the following attribute to a symbol or its output net:

LOC=FB*nn***[**_*mm***]**

where *nn* is a valid function block number and *mm* (optional) is a valid macrocell number for the target device.

Prohibiting the Use of Device Pins

The PROHIBIT attribute allows you to reserve device pins for later use, or simply prevent them from being used at all. For instance, if you anticipate design changes in the future and want to set traces on your printed circuit board now, you can use PROHIBIT to prevent the fitter from using pins associated with those traces. Then, when you decide to use the traces, you can use the LOC attribute to assign those pins to new input/output buffers you place in your design. To use PROHIBIT, instantiate a CONFIG symbol and attach the PROHIBIT attribute to it. The syntax is as follows:

PROHIBIT=P*nn***[,P***nn***]...**

where *nn* is the pin number for PC, PQ and VQ packages, and *rc* (row,column) for BG or PG packages.

In the Constraints Editor, **Prohibit I/O Locations** prevents all selected I/O pins from being used by the design. This dialog can be entered using a dialog box provided in the **Ports** tab.

Pin Assignment Precautions

You can apply the LOC and PROHIBIT attributes 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 combinatorial 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 combinatorial 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 20. 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 in Design Manager

In Design Manager, 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 **`Advanced`**
4. Place a value in the box adjacent to **`Collapsing Pterm Limit`**.

The allowable range for the p-terms parameter is between 5 and 90; the default is 20.

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, 30 or 35 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.

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 control attribute KEEP to specific nodes in your design.

Applying the following attribute to a logic symbol or net in the middle of a logic function prevents collapsing of that logic node into its fan-outs:

KEEP

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

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

When the KEEP attribute is placed on a symbol, it inhibits logic optimization on all macrocells used to implement the symbol. For example, if you place KEEP on a macro symbol (like D2_4E), all outputs and internal nodes of the decoder will be prevented from collapsing. This is usually not desirable.

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

Forcing Collapsing of a Logic Node

You can also force a logic symbol to collapse into all of its fanouts by placing the following attribute on the symbol or its output net:

COLLAPSE

The collapse attribute affects all logic functions contained within a symbol. If you want to force collapsing of a multi-symbol logic chain, you may need to use multiple collapse attributes.

Multilevel Logic Optimization

Multilevel Logic Optimization seeks to simplify the total number of logic expressions in a design, and then collapse the logic in order to meet user objectives such as density, speed and timespecs. This optimization targets CPLD architecture, making it possible to collapse to the macrocell limits, reduce levels of logic, and minimize the total number of pterms.

Multilevel Logic Optimization extracts combinatorial logic from your design. Combinatorial logic includes:

- register-to-register logic
- path-to-register logic
- register-to-path logic
- path-to-path logic

Multilevel Logic Optimization operates on combinatorial logic according to the following rules:

1. If timespecs are set, the program will optimize for speed to meet timespecs.
2. If timespecs are not set, the program will optimize either for speed or density, depending on the user setting of **Timing Optimization**.
   a) If **Timing Optimization** is turned on, the combinational logic will be mapped for speed.
   b) If **Timing Optimization** is turned off, the combinational logic will be mapped for density. The goal of optimization will then be to reduce th total number ot pterms.
3. Logic marked with the attribute **NOREDUCE** will not be extracted or optimized.

Setting Multilevel Logic Optimization

Multilevel Logic Optimization can be set from the **Advanced** tab of the **Implementation Options** template of the Design Manager as follows:

1. Select **Design** → **Implement**
2. Press the **Options** softkey.
3. Select **Edit Template**
4. Select the **Advanced** tab.
5. Place a check in the **Use Multilevel Logic Optimization** box (the default is **On**).

Multilevel Logic Optimization will operate when you run the fitter.

Controlling Timing Paths

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

- Global Timing Optimization
- Timing Constraints (Timespecs)

Timing Optimization

When you create a new project in the Design Manager, it is configured by default to optimize primarily for speed rather than for density. The **Optimize Speed** template containing all the recommended fitter options to achieve best first-run speed is selected in the **Options** menu. With the **Optimize Speed** template selected, 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.

If you want to optimize for density instead of speed, select the **Optimize Density** template in the Design Manager Options menu. In the **Optimize Density** template, Timing Optimization is turned off. If you are customizing your own template settings, you can directly 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 tab:

   Advanced

4. Lastly, place a check in the **Off** box adjacent to **Use Timing Optimization**.

Timing Specifications

This section describes the basic methods for entering timing specifications.

The software provides a set of timing constraint commands that you can use to specify the timing requirements of your Xilinx design. After compiling your design, the Xilinx software reads both your design netlist and your timing constraint commands and performs timing optimization according to your specifications.

The following path types can be controlled using timing specifications:

Pad-to-pad delay	Input port to an output port
Register setup time	Input port to the data pin of a flip-flop, including flip-flop setup requirements
Register-to-register	Clock pin of a flip-flop to the data pin of the same or different flip-flop, including flip-flop setup requirements
Clock-to-output delay	Clock pin of a flip-flop to an output port

This section outlines the commands that you can use to create timing specifications for your Xilinx CPLD designs.

Entering Timing Specifications using the Constraints Editor

This following tells you how to create new timing constraints from the Global tab window, and the Ports tab window of the Constraints Editor.

Note: To start the Contraints Editor from the Design Manager, simply click on the Contraints Editor Icon found on the right side of the Design Manager graphical interface. See the *Constraints Editor Guide* for more information.

Note: After a constraint is created, it appears syntactically in the Editable Constraints list the same way it appears in the UCF.

From the Global Tab Window

The following global constraints are created from the Global tab window of the Constraints Editor.

- "Clock Period"
- "Pad to Setup"
- "Clock to Pad"
- "Pad to Pad"

Clock Period

To create a new clock period constraint, open the Global tab window. Follow the steps below.

1. In the Period column and in the row associated with the appropriate clock net name, double-click the left mouse button. This opens the Clock Period dialog box.

Note: An alternate way to invoke the Clock Period dialog box: right click in any row or column (except the heading), then click **`Period`** from the pop-up window.

Note: By default, a timing specification name for the clock period is displayed in the Time Spec text box. The name consists of the letters "TS" and an identifier in the form of the clock net name; for example, TS_CLK1. You may change the timing specification name as desired. If the name you choose does not begin with "TS", the program will prepend it to the name.

2. In the Clock Period field, you may signify an explicit period for a clock or you may designate a period which is relative to a clock period defined in another timing specification.
3. If you select Explicit, enter a value in the Time text box and select a unit from the Units pull-down list. Select **`Start High`** or **`Start Low`** for the initial clock pulse, enter a value in the Time HIGH or Time LOW text box, and a value for time high or low from the pull-down list. The default for Units is %, which is the high or low percentage of the total.
4. If you select **`Relative to other Period Time Spec`**, select the name of the time spec in the Reference Time Spec text box,

select **Multiply by** or **Divide by** and a value in the Factor text box.

5. Click **OK**.

Pad to Setup

Pad to Setup creates a constraint which allows you to specify the timing relationship between an external clock and data at the pins of a device. The Global tab window generates a dialog box from which you can specify a pad to setup requirement for all inputs that are clocked by the clock net identified by the user.

1. In the Pad to Setup column and in the row associated with the appropriate clock net name, double-click the left mouse button. This opens the Pad to Setup dialog box.

Note: An alternate way to invoke the Pad to Setup dialog box: right click in any row or column (except the heading), then click **Pad to Setup** from the pop-up window.

2. Enter a time requirement in the Time Requirement text box and the desired unit from the pull-down list.

Note: The Relative to Clock Net field contains the name that you selected in Step 1. You are not allowed to change this field.

3. Click **OK**.

Clock to Pad

This creates a constraint which allows you to specify the timing relationship between an external clock and data at the pins of a device. The Global tab window generates a dialog box from which you can specify a time requirement for a global clock.

1. In the Clock to Pad column and in the row associated with the appropriate clock net name, double-click the left mouse button. This opens the Clock to Pad dialog box.

Note: An alternate way to invoke the Clock to Pad dialog box: right click in any row or column (except the heading), then click **Clock to Pad** from the pop-up window.

2. Enter a time requirement in the Time Requirement text box and the desired unit from the pull-down list.

Note: The Relative to Clock Net field contains the name that you selected in Step 1. You are not allowed to change this field.

3. Click **OK**.

Pad to Pad

1. Enter a time value in the Pad to Pad text box, then select the desired unit from the Units pull-down list.
2. Click **File** → **Save**.

From the Ports Tab Window

The following constraints are created from the Ports tab window of the Constraints Editor.

- "Pad to Setup"
- "Clock to Pad"

Pad to Setup

The Ports tab window generates a dialog box from which you can specify a time requirement for individual ports relative to a selected clock net.

1. In the Pad to Setup column and in the row associated with the appropriate port name, double-click the left mouse button. This opens the Pad to Setup dialog box.

Note: An alternate way to invoke the Pad to Setup dialog box: right click in any row or column (except the heading), then click **Pad to Setup** from the pop-up window.

2. Enter a time requirement in the Time Requirement text box and the desired unit from the pull-down list.
3. Select the clock that clocks this input from the Relative to Clock Net pull-down list.
4. Click **OK**.

Clock to Pad

The Ports tab window generates a dialog box from which you can specify a time requirement for a port relative to a selected clock net.

1. In the Clock to Pad column and in the row associated with the appropriate port name, double-click the left mouse button. This opens the Clock to Pad dialog box.

Note: An alternate way to invoke the Clock to Pad dialog box: right click in any row or column (except the heading), then click **Clock to Pad** from the pop-up window.

2. Enter a time requirement in the Time Requirement text box and the desired unit from the pull-down list.
3. Select the clock that clocks this input from the Relative to Clock Net pull-down list.
4. Click **OK**.

Slow/Fast Path Exceptions (FROM/THRU/TO)

This generates the FROM / THRU / TO dialog box, which allows you to specify an explicit maximum delay between groups of elements and through intermediate points.

This is particularly useful for defining pad-to-pad propagation delays between specific device pins.

1. Click **Specify** in the Slow / Fast Path Exceptions (FROM / THRU / TO) field. This opens the FROM / THRU / TO dialog box.
2. In the Time Spec text box enter a time specification name. The name should be entered in the form TSid, where id is a unique name. The name can consist of letters, numbers, or the underscore character (_). If you do not enter TS, the software will automatically place it before your id
3. Make a selection from the From Group and To Group pull-down lists (All, All FFS, All PADS).
4. Enter a value for time in the Time text box and select the appropriate unit from the Units pull-down list.

- Click **OK**.

Entering Timing Specifications in a Schematic

The TIMESPEC schematic primitive, as illustrated in the “TIMESPEC Primitive” figure, serves as a placeholder for timing specifications, which are called TS attribute definitions. Every TS attribute must be

defined in a TIMESPEC primitive. Every TS attribute begins with the letters "TS" and ends with a unique identifier that can consist of letters, numbers, or the underscore character (_).

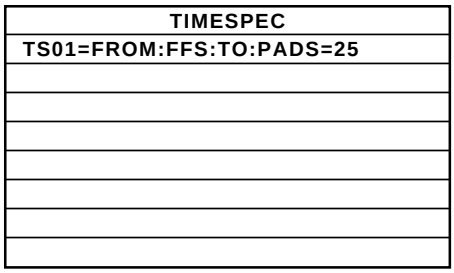

Figure 3-2 TIMESPEC Primitive

A TS attribute defines the allowable delay for paths in your design. The basic syntax for a TS attribute is:

TS*identifier***=FROM:***source_group***:TO:***dest_group***:***delay*

where **TS***identifier* is a unique name for the TS attribute, *source_group* and *dest_group* are groups of start points and end points, and *delay* defines the maximum delay for the paths between the start points and end points. The *delay* parameter defines the maximum delay for the attribute. Nanoseconds are the default units for specifying delay time in TS attributes. You can also specify delay using other units, such as picoseconds or megahertz.

Note: Keywords, such as FROM, TO, and TS appear in the documentation in upper case; however, you can enter them in the TIMESPEC primitive in either upper or lower case. You cannot enter them in a combination of lower and upper case.

The basic TS attribute is described in detail in the "Basic TIMESPEC Syntax" section. More detailed forms of the attribute are also described in that section.

Entering Timing Specifications in a Constraints File

You can enter timing specifications as constraints in a UCF file. When you then run the fitter on your design, your timing specifications will be added to the design database as part of the NGD file.

The basic syntax for a timing specification entered in a constraints file is the TS attribute syntax described in the “Basic TIMESPEC Syntax” section.

Specifying Groups in TS Attributes

In a TS attribute, you specify the set of paths to be analyzed by grouping start and end points in one of the following ways:

- You can refer to a predefined group by specifying one of the corresponding keywords — FFS or PADS.
- You can create your own groups within a predefined group by tagging pad or flip-flop symbols with TNM (timing name) attributes.
- You can create groups that are combinations of existing groups using TIMEGRP symbols.
- You can create groups by pattern matching on net names.

The following sections discuss each method in detail.

Using Predefined Groups

You can refer to a group of flip-flops or pads by using the corresponding keywords.

Keyword	Description
FFS	Macrocell or IOB flip-flops, including those used to implement transparent latch macros.
PADS	Input/output pads

Timing specification From-To statements enable you to define timing specifications for paths between predefined groups. The following examples are TS attributes attached to a TIMESPEC primitive or are entered in the UCF. This method enables you to easily define default timing specifications for the design, as illustrated by the following examples:

```
TS01=FROM:FFS:TO:FFS:30
TS04=FROM:FFS:TO:PADS:55
```

A predefined group can also carry a name qualifier; the qualifier can appear any place where the predefined group is used. This name qualifier restricts the number of elements being referred to. The syntax used is as follows:

predefined group **(***name_qualifier* **[:** *name_qualifier* **])**

where *name_qualifier* is the full hierarchical name of the net that is sourced by the primitive being identified.

The name qualifier can include wildcard characters (* to indicate any number of characters or ? to indicate a single character) which allows the specification of more than one net or allows you to shorten the full hierarchical name to something that is easier to type.

Creating User-Defined Groups Using TNMs

A TNM (timing name) is an attribute that can be used to identify the elements that make up a group of end-points (pads and flops) which can then be used in a timing specification. A TNM is a flag that you place directly on elements in your schematic to identify specific flops and pads. All symbols tagged with the same TNM value are considered members of the same group. Place TNM attributes directly on your schematic using the following syntax:

TNM=*identifier*

where *identifier* is a group name that consists of any combination of letters, numbers, or underscores. Keep the TNM short for convenience and clarity.

Warning: Do not use the reserved words FFS, LATCHES, PADS, RAMS, RISING, FALLING, TRANSHI, TRANSLO, or EXCEPT, as *identifier*s.

You can specify as many groups of end points as are necessary to describe the performance requirements of your design. However, to simplify the specification process and reduce the place-and-route time, use as few groups as possible.

You can attach a TNM attribute to a macro symbol, in which case the TNM is applied to all applicable primitives inside the macro.

A predefined group can be used to qualify the application of a TNM attribute using the following syntax:

TNM=*predefined_group***:***identifier*

where *predefined_group* is one of the groups, FFS or PADS, and *identifier* is any valid group name. Paths defined by the TNM are traced forward, through any number of gates or buffers, until they reach a member of the specified *predefined_group*. That element is added to the specified TNM group. This mechanism is called forward tracing.

A predefined group in a TNM statement can have a net name qualifier (for example, TMM=FFS:(FRED*):GRP_A), as described in the "Creating Groups by Pattern Matching" section. In this example, the TNM is forward traced until it finds all flops producing signals matching the name FRED*; each such flop is then added to the group name GRP_A.

You can use several methods for tagging groups of end points: placing identifiers on nets, macro or primitive pins, primitives, or macro symbols. Which method you choose depends on how the path end points are related in your design. For each of these elements, you can use the predefined group syntax described earlier in this section.

The following subsections discuss the different methods of placing TNMs in your design. The same TNM attribute can be used as many ways and as many times as necessary to get the TNM applied to all of the elements in the desired group.

You can place TNM attributes in either of two places: in the schematic as discussed in this section or in a constraints file (UCF or NCF). The syntax for specifying TNMs in a UCF or NCF constraints file is described in the "Attributes" appendix.

Placing TNMs on Nets

The TNM attribute can be placed on any net in the design. The attribute indicates that the TNM value should be attached to all valid elements fed by all paths that fan forward from the tagged net. Forward tracing stops at any flip-flop or pad.

Note: A TNM placed on a net connected to the output of a flip-flop does not apply to that flip-flop, but is instead forward-traced to subsequent flops or pads.

Placing TNMs on Macro or Primitive Pins

The TNM attribute can be placed on any macro or component pin in the design if the design entry package allows placement of attributes on macro or primitive pins. The attribute indicates that the TNM value should be attached to all valid elements fed by all paths that fan forward from the tagged pin. Forward tracing stops at any flip flop or pad.

Placing TNMs on Primitive Symbols

You can group individual logic primitives explicitly by flagging each symbol, as illustrated by the "TNM on Primitive Symbols" figure.

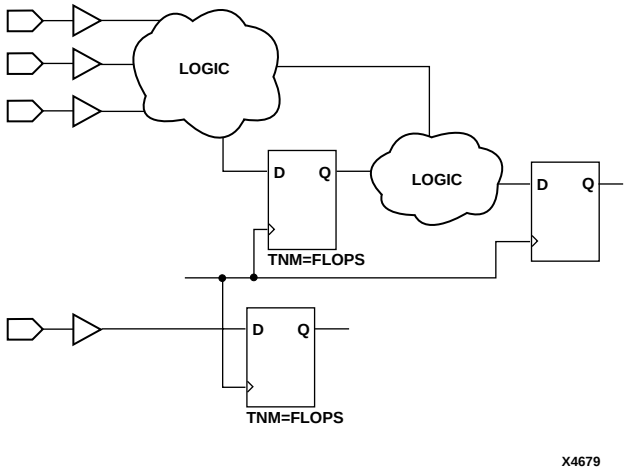

Figure 3-3 TNM on Primitive Symbols

In the "TNM on Primitive Symbols" figure, the flip-flops tagged with the TNM form a group called "'FLOPS." The untagged flip-flop is not part of the group.

Place only one TNM on each symbol, load pin, or macro load pin. If you want to assign more than one identifier to the same symbol, include all identifiers on the right side of the equal sign (=) separated by a semicolon (;), as follows:

```
TNM=joe;fred
```

Placing TNMs on Macro Symbols

A TNM attribute attached to a macro symbol (either a library or user macro) indicates that all applicable elements inside the macro (at all levels of hierarchy below the tagged macro) are part of the named group.

When a macro contains more than one symbol type and you want to group only a single type, use the TNM identifier in conjunction with one of the predefined groups: FFS or PADS, as indicated by the following syntax examples:

```
TNM=FFS:identifier
TNM=PADS:identifier
```

If you want to place an identifier on more than one symbol type, separate each symbol type and identifier with a semicolon (;) as illustrated by the following example:

```
TNM=FFS:FLOPS;PADS:OPADS
```

If multiple symbols of the same type are contained in the same hierarchical block, you can simply flag that hierarchical symbol, as illustrated by the “TNM on Macro Symbol” figure. In the figure, all flip-flops included in the macro are tagged with the TNM ‘‘FLOPS.” By tagging the macro symbol, you do not have to tag each underlying symbol individually.

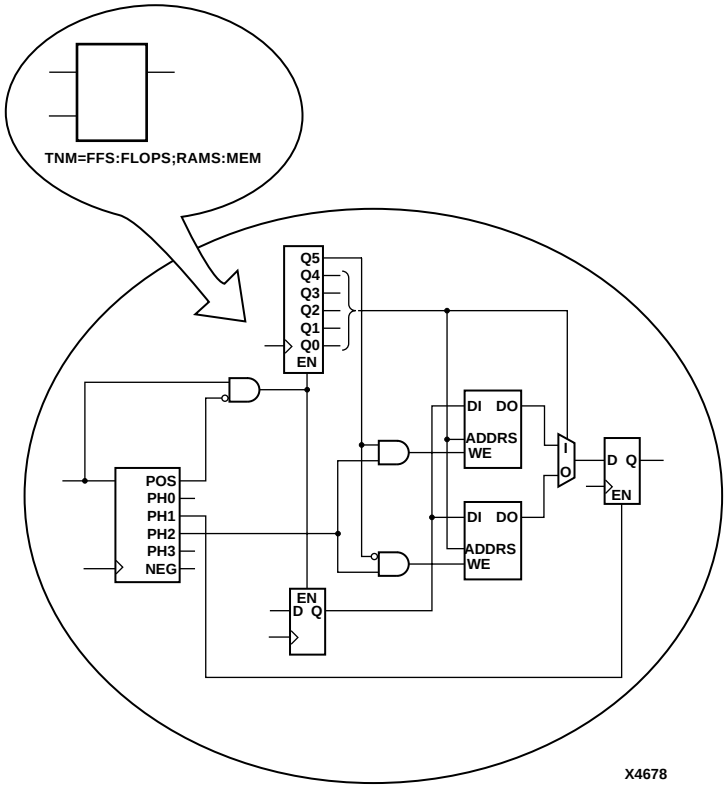

Figure 3-4 TNM on Macro Symbol

Placing TNMs on Nets or Pins to Group Flip-Flops

You can easily group flip-flops by flagging a common input net, typically either a clock net or an enable net. If you attach a TNM to a net or load pin, that TNM applies to all flip-flops that are reached through the net or pin. That is, that path is traced forward, through any number of gates or buffers, until it reaches a flip-flop. That element is added to the specified TNM group.

Placing a TNM on a net is equivalent to placing that TNM attribute on every load pin of the net. Use pin TNM attributes when you need finer control.

The “TNM on Net Used to Group Flip-Flops” figure illustrates the use of a TNM on a net that traces forward to create a group of flip-flops. In the figure, the attribute TNM=FLOPS traces forward to the first two flip-flops, which form a group called FLOPS. The bottom flip-flop is not part of the group FLOPS

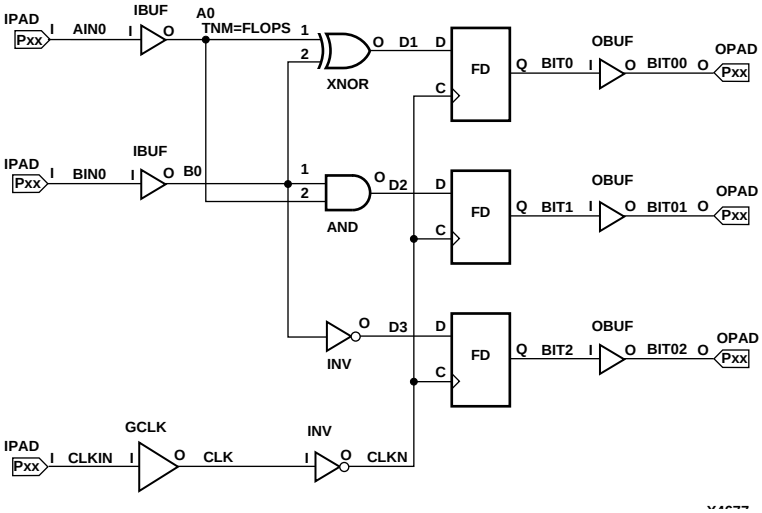

Figure 3-5 TNM on Net Used to Group Flip-Flops

The “TNM on Clock Pin Used to Group Flip-Flops” figure illustrates placing a TNM on a clock net, which traces forward to all three flip-flops and forms the group Q_FLOPS:

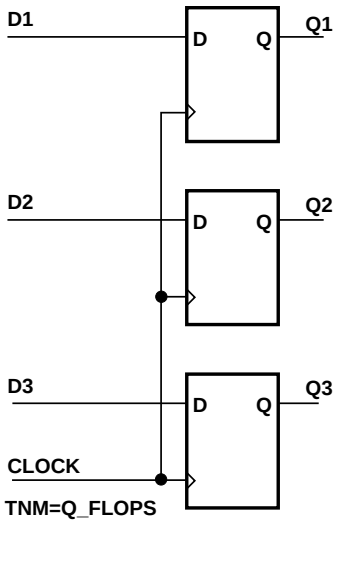

Figure 3-6 TNM on Clock Pin Used to Group Flip-Flops

The TNM parameter on nets or pins is allowed to have a qualifier.

For example:

```
TNM=FFS:data
```

A qualified TNM is traced forward until it reaches the first flip-flop. If that flip-flop matches the qualifier, the flip-flip is given that TNM value. Whether or not there is a match, the TNM is *not* traced through that flip-flop.

Creating New Groups from Existing Groups

In addition to naming groups using the TNM identifier, you can also define groups in terms of other groups. You can create a group that is a combination of existing groups by defining a TIMEGRP attribute as follows:

newgroup=*existing_grp1*:*existing_grp2* [:*existing_grp3* . . .]

where *newgroup* is a newly created group that consists of existing groups created via TNMs, predefined groups, or other TIMEGRP attributes.

Mentor Users — You must specify a leading equal sign (=) when defining TIMEGRP attributes, for example, =newgroup. The preceding equal sign lets Mentor know that this is a user-defined attribute. Refer to the *Mentor Graphics Interface/Tutorial Guide* for more information.

TIMEGRP attributes reside in the TIMEGRP primitive, as illustrated in the "TIMEGRP Primitive" figure. Once you create a TIMEGRP attribute definition within a TIMEGRP primitive, you can use it in the TIMESPEC primitive. Each TIMEGRP primitive can hold up to eight group definitions. Since your design might include more than eight TIMEGRP attributes, you can use multiple TIMEGRP primitives.

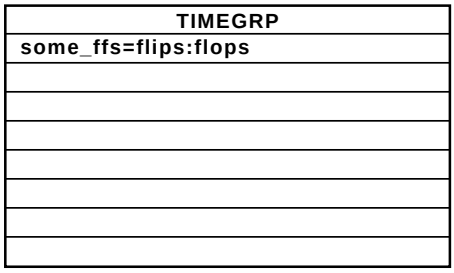

Figure 3-7 TIMEGRP Primitive

You can place TIMEGRP attributes in either of two places: in the TIMEGRP primitive on the schematic as discussed in this section or in a constraints file (UCF or NCF). The syntax for specifying TNMs in a UCF or NCF constraints file is described in the "Attributes" appendix.

You can use TIMEGRP attributes to create groups using the following methods:

- Combining multiple groups into one
- Creating groups by exclusion

- Defining flip-flop subgroups by clock sense

The following subsections discuss each method in detail.

Combining Multiple Groups into One

You can define a group by combining other groups. The following syntax example illustrates the simple combining of two groups:

```
big_group=small_group:medium_group
```

In this syntax example, small_group and medium_group are existing groups defined using a TNM or TIMEGRP attribute. Within the TIMEGRP primitive, TIMEGRP attributes can be listed in any order; that is, you can create a TIMEGRP attribute that references another TIMEGRP attribute that appears after the initial definition.

Warning: A circular definition, as shown below, causes an error when you run your design through NGDBUILD.

```
many_ffs=ffs1:ffs2
ffs1=many_ffs:ffs3
```

Creating Groups by Exclusion

You can define a group that includes all elements of one group except the elements that belong to another group, as illustrated by the following syntax examples:

group1=*group2*:**EXCEPT**:*group3*

- *group1* represents the group being defined. It will contain all of the elements in *group2* except those that are also in *group3*.
- *group2* and *group3* can be a valid TNM, predefined group, or TIMEGRP attribute.

As illustrated by the following example, you can specify multiple groups to include or exclude when creating the new group.

group1=*group2*:*group3*:**EXCEPT**:*group4*:*group5*

The example defines a *group1* that includes the members of *group2* and *group3*, except for those members that are part of *group3* or *group4*. All of the groups before the keyword EXCEPT are included, and all of the groups after the keyword are excluded.

Certain reserved words cannot be used as group names. These reserved words are described in the “Creating User-Defined Groups Using TNMs” section.

Defining Flip-Flop Subgroups by Clock Sense

You can create subgroups using the keywords RISING and FALLING to group flip-flops triggered by rising and falling edges.

group1=RISING:ffs
group2=RISING:*ffs_group*
group3=FALLING:ffs
group4=FALLING:*ffs_group*

where *group1* to *group4* are new groups being defined. The *ffs_group* must be a group that includes only flip-flops.

Note: Keywords, such as EXCEPT, RISING, and FALLING, appear in the documentation in upper case; however, you can enter them in the TIMESPEC primitive in either lower or upper case. You cannot enter them in a combination of lower and upper case.

The following example defines a group of flip-flops that switch on the falling edge of the clock.

```
falling_ffs=FALLING:ffs
```

Creating Groups by Pattern Matching

When creating groups, you can use wildcard characters to define groups of symbols whose associated net names match a specific pattern.

How to Use Wildcards to Specify Net Names

The wildcard characters, * and ?, enable you to select a group of symbols whose output net names match a specific string or pattern. The asterisk (*) represents any string of zero or more characters. The question mark (?) indicates a single character.

For example, DATA* indicates any net name that begins with “DATA,” such as DATA, DATA1, DATA22, DATABASE, and so on. The string NUMBER? specifies any net names that begin with “NUMBER” and end with one single character, for example, NUMBER1, NUMBERS but not NUMBER or NUMBER12.

You can also specify more than one wildcard character. For example, *AT? specifies any net names that begin with any series of characters followed by "AT" and end with any one character such as BAT1, CAT2, and THAT5. If you specify *AT??, you would match BAT11, CAT26, and THAT50.

Pattern Matching Syntax

The syntax for creating a group using pattern matching is shown below:

group=*predefined_group*(*pattern*)

where *predefined_group* can only be one of the following predefined groups—FFS or PADS. The *pattern* is any string of characters used in conjunction with one or more wildcard characters.

For flip-flops specify the output net name. For pads, specify the name of the external net connected to the pad.

The following example illustrates creating a group that includes the flip-flops that source nets whose names begin with $1I3/FRED.

```
group1=ffs($1I3/FRED*)
```

The following example illustrates a group that excludes certain flip-flops whose output net names match the specified pattern:

```
this_group=ffs:EXCEPT:ffs(a*)
```

In this example, this_group includes all flip-flops except those whose output net names begin with the letter "a."

Additional Pattern Matching Details

In addition to using pattern matching when you create timing groups, you can specify a predefined group qualified by a pattern any place you specify a predefined group. The syntax below illustrates how pattern matching can be used within a timing specification:

TS*identifier*=**FROM:***predefined_group*(*pattern*)**:TO:***predefined_group* (*pattern*)**:***delay*

Instead of specifying just one pattern, you can also specify a list of patterns separated by a colon (:) as illustrated below:

```
some_ffs=ffs(a*:b?:c*d)
```

The group `some_ffs` contains flip-flops whose output net names:

- Start with the letter “a”

  or

- Contain two characters; the first character is “b”

  or

- Start with “c” and end with “d”

Basic TIMESPEC Syntax

Within the TIMESPEC primitive, you use the following syntax to specify timing requirements between specific end points:

TS*identifier***=FROM:***source_group***:TO:***dest_group***:***delay*

The From-To statements are TS attributes that reside in the TIMESPEC primitive. The parameters *source_group* and *dest_group* must be one of the following:

- predefined groups
- previously created TNM identifiers
- groups defined in TIMEGRP symbols

Predefined groups consist of FFS or PADS and are discussed in the “Using Predefined Groups” section. TNMs are introduced in the “Creating User-Defined Groups Using TNMs” section. TIMEGRP symbols are introduced in the “Creating New Groups from Existing Groups” section.

Note: Keywords, such as FROM, TO, and TS appear in the documentation in upper case; however, you can enter them in the TIMESPEC primitive in either upper or lower case. You cannot enter them in a combination of lower and upper case.

The *delay* parameter defines the maximum delay for the attribute. Nanoseconds are the default units for specifying delay time in TS attributes. You can also specify delay using other units, such as picoseconds or megahertz. Refer to the “Specifying Time Delay in TS Attributes” section later in this chapter for more information on time delay.

The following examples illustrate the use of From-To TS attributes:

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

You can place TS attributes containing From-To statements in either of two places: in the TIMESPEC primitive on the schematic as discussed in this chapter or in a constraints (UCF) file. See the "Attributes" appendix for more information about specifying timing requirements in a constraints file.

Specifying Time Delay in TS Attributes

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:

- PS for picoseconds, NS for nanoseconds, US for microseconds, or MS for milliseconds
- MHZ for megahertz or KHZ for kilohertz

As an alternate way of specifying time delay, you can specify one time delay in terms of another. This method is described in the next section.

Specifying a TS Attribute Delay in Terms of Another

Instead of specifying a time or frequency in a TS attribute definition, you can specify a multiple or division of another TS attribute. This is useful in a system where all clocks are derived from a master clock; in this situation, changing the timing specification for the master clock changes the specification for all clocks in the system.

Use the syntax below to specify a TS attribute delay in terms of another.

TS*identifier***=***specification***:***reference_TS_attribute***[*|/]***number*

where *number* can be either a whole number or a decimal. The specification can be any From-To statement as illustrated by the following examples:

```
FROM:PADS:TO:PADS
FROM:group1:TO:group2
FROM:tnm_identifier:TO:FFS
```

Use "*" to represent multiplication and "/" to represent division. The specification type of the reference TS attribute does not need to be the same as the TS attribute being defined; however, it must not be specified in terms of AUTO or IGNORE.

Setting TIMESPEC Priorities

There may be situations where two TIMESPECs at the same level of priority conflict. In these cases you can define the priority of a TIMESPEC using the following syntax:

normal_timespec_syntax : **PRIORITY :** *integer*

where *normal_timespec_syntax* is a legal TIMESPEC and *integer* represents the priority (the smaller the number, the higher the priority). The number can be positive, negative, or zero, and the value only has meaning when compared with other PRIORITY values.

See the "Controlling Timing Paths" for more details.

Defining a Clock Period

A clock period specification checks timing clocked by the net (all paths that terminate at a register clocked by the specified net). The period specification is attached to the clock net. The definition of a clock period is unlike a FROM:TO style specification, because the timing analysis tools will automatically take into account any inversions of the clock net at register clock pins.

A PERIOD constraint on the clock net would generate a check for delays on all paths that terminate at a pin that has a setup or hold timing constraint relative to the clock net. This could include the data paths DI to MC1.D, MC1.Q to MC2.D, as well as the paths D0 to MC1.R and EN to MC2.EC (if the reset/enable were synchronous with respect to the clock).

Simple Method

A simple method of defining a clock period is to attach the following attribute directly to a net in the path that drives the register clock pins:

PERIOD = *period* **: { HIGH | LOW }: [***high_or_low_time***]**

where *period* is the required clock period. The default units are nanoseconds, but the timing number can be followed by ps, ns, us, or ms. Units may be entered with or without a leading space, and are case-insensitive. The *high_or_low_time* is the optional high or low time depending on the **HIGH|LOW** keyword. If an actual time is specified, it must be less than the period. If no high or low time is specified the

default duty cycle is 50%. The default units for *high_or_low_time* is ns, but the number can be followed by %, ps, us or ms.

The PERIOD constraint is forward traced in exactly the same way a TNM would be and attaches itself to all of the flip flops that the forward tracing reaches. There are no rules about not tracing through certain elements. If a more complex form of tracing behavior is required (for example, where gated clocks are used in the design), you must place the PERIOD on a particular net, or use the preferred method described next.

Preferred Method

The preferred method for defining a clock period allows more complex derivative relationships to be defined as well as a simple clock period. The following attribute is attached to a TIMESPEC symbol in conjunction with a TNM attribute attached to the relevant clock net.

TS*identifier***=PERIOD:** *TNM_reference***:** *period***: {HIGH | LOW}: [***high_or_low_time***]**

where *identifier* is a reference identifier that has a unique name, *TNM_reference* is the identifier name that is attached to a clock net (or a net in the clock path) using a TNM attribute, and *period* is the required clock period. The default units for *period* are nanoseconds, but the number can be followed by ps, ns, us, or ms. Units may be entered with or without a leading space, and are case-insensitive. *High_or_low_time* is the optional high or low time depending on the **HIGH|LOW** keyword. If an actual time is specified, it must be less than the period. If no high or low time is specified the default duty cycle is 50%. The default units for *high_or_low_time* is ns, but the number can be followed by %,, ps, ns, or mst.

Specifying Setup Time Using the OFFSET Constraint

To specify setup time using the **OFFSET** constraint, use the following syntax:

NET *input_pad* **OFFSET=IN:***delay***:BEFORE:***clock_pad***;**

OFFSET=IN:*delay***:BEFORE:***clock_pad;*

where *delay* is in nanoseconds.

Specifying Clock-to-Output Delay Using the OFFSET Constraint

To specify clock-to-output delay using the **OFFSET** constraint, use the following syntax:

NET *output_pad* **OFFSET=OUT:***delay***:AFTER:***clock_pad***;**

OFFSET=OUT:*delay***:AFTER:***clock_pad***;**

where *delay* is in nanoseconds.

Constraints Priority

In some cases, two timing specifications will cover the same path. For cases where the two timing specifications are mutually exclusive, the following constraint rules apply:

- Priority depends on the file in which the constraint appears. A constraint in a file accessed later in the design flow replaces a constraint in a file accessed earlier in the design flow. Priority is as follows (first listed is the highest priority, last listed is the lowest):
  - Constraints in a Netlist Constraints File (NCF)
  - Constraints in a User Constraints File (UCF)
  - Attributes in a schematic
- If two timing specifications cover the same path, the priority is as follows (first listed is the highest priority, last listed is the lowest):
  - FROM:TO specifications
  - PERIOD specifications
- FROM:TO statements have a priority order that depends on the type of source and destination groups included in a statement. The priority is as follows (first listed is the highest priority, last listed is the lowest):
  - Both the source group and the destination group are user-defined groups
  - Either the source group or the destination group is a predefined group

- Both the source group and the destination group are predefined groups

If two constraints are in the same category, the user-defined priority described in the“Setting TIMESPEC Priorities” section is used to determine which constraint takes precedence.

Reducing Levels of Logic

The XC9000 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. The “Reducing Levels of Logic” figure shows two functionally similar designs, one that is efficient for CPLD architectures and one that is inefficient.

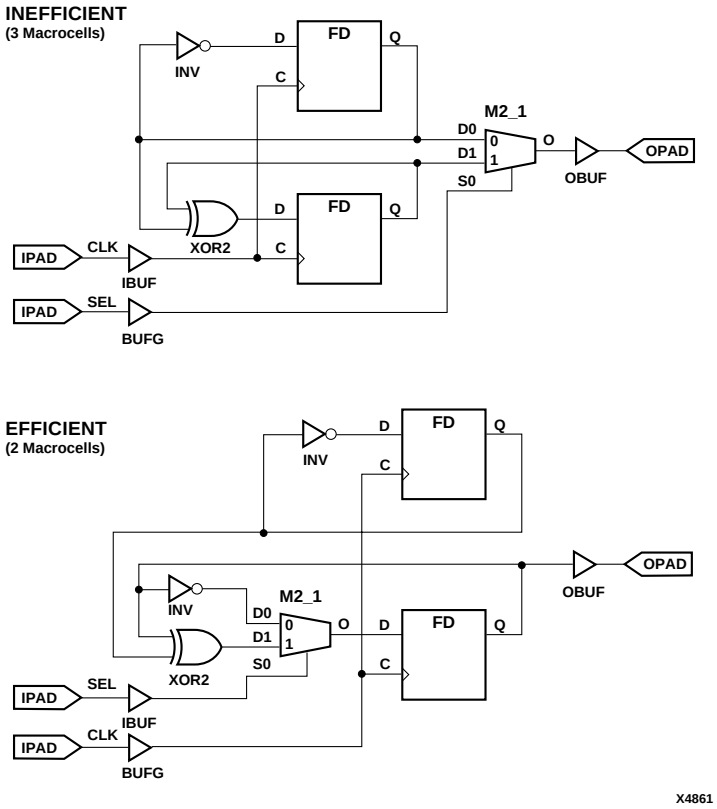

Figure 3-8 Reducing Levels of Logic

Controlling XC9500 Local Feedback Routing

By default, all internal nodes in an XC9500 design (those that remain after collapsing) are routed via the FastCONNECT structure. There are also higher-speed routing paths that feed back from each macrocell to the inputs of the same local function block. To use the local feedback path for a particular node in your design, both the source logic and the load logic on the node must be mapped to the same function block. If you entered timespecs for your design, the fitter will attempt to use local feedback where necessary to satisfy your timespecs. When the feedback path between two macrocells is

involved in a timing-constrained path, and if the timing constraints cannot be met using FastCONNECT routing, the fitter will attempt to map both macrocells to the same function block and use the local feedback path.

If you want to explicitly control the use of local feedback routing, you must:

- Constrain both the driving function and load function(s) to the same function block using the LOC=FB*nn* attribute.
- Apply a timing specification to the path that would require the local feedback path (so that the path cannot be satisfied using FastCONNECT routing delays).

  Hint: You can specify the value of 1 ns in your timespec to tell the fitter to use local feedback, even though the fitter will warn you that it cannot satisfy your timespecs.

As an alternative to applying a timing specification, you can turn on the **`Use Local Macrocell Feedback`** option in the Design Manager. But, that would allow the local feedback path to be used for any other internal nodes in the design that run between two functions that happen to get mapped to the same function block.

Note: Turning off the **`Use Local Macrocell Feedback`** option does not prevent the fitter from automatically using the local feedback path when necessary to satisfy your timespecs.

Note: The XC9536 device does not have local feedback.

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. This chapter contains the following sections:

- “Read-Back Registers”
- “Bidirectional Signals and Buses”
- “Multiplexing Tristate Signals”
- “Combinatorial Feedback Loops”

Read-Back Registers

The “Read-Back Register Example” figure 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.

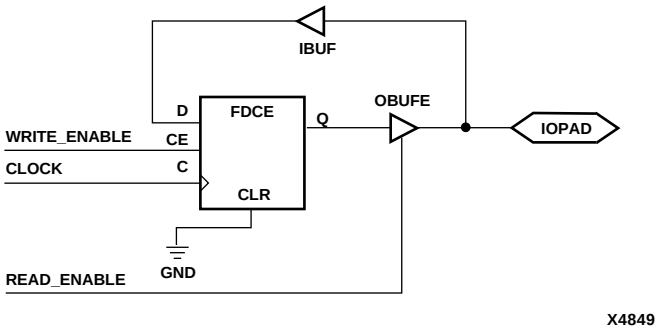

Figure 4-1 Read-Back Register Example

Bidirectional Signals and Buses

The “Bidirectional Signals and Buses” figure, part A, shows how to specify a bidirectional pin. Part B shows that you can have a bidirectional signal passing through the chip. To make a bidirectional bus, use bus components as shown in part C.

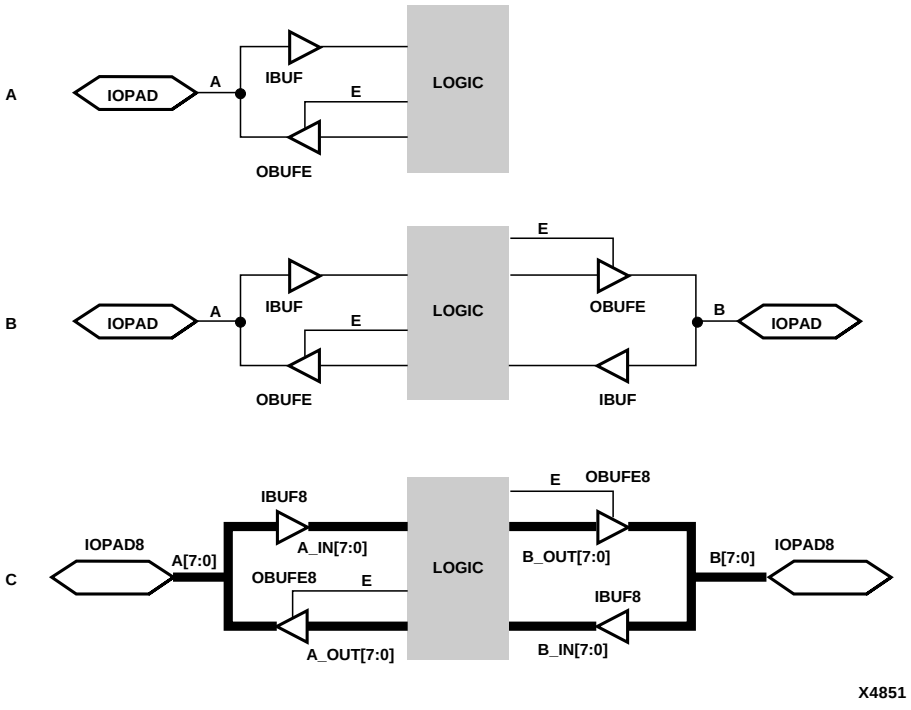

Figure 4-2 Bidirectional Signals and Buses

Multiplexing Tristate Signals

Note: XC9500 devices can emulate tristate bussing using special gates that disable the macrocell feedback path to the FastCONNECT matrix. XC9500XL and XC9500XV devices do not support internal tristate buffers. Do not use BUFE or BUFT components in XC9500XL or XC9500XV designs.

Three methods of multiplexing tristate signals are shown in the "Methods of Multiplexing Tristate Signals" figure. 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 the “Methods of Multiplexing Tristate Signals” figure, part A, uses tristate 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 not available to any other on-chip functions because the internal feedbacks are tristated; therefore counters will not work with Method A, but will work with Method C.

Method B, shown in the “Methods of Multiplexing Tristate Signals” figure, part B, 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 the “Methods of Multiplexing Tristate Signals” figure, part C, uses a multiplexer instead of tristate buffers. This method results in a longer clock-to-out delay than method B. To shorten the clock-to-output delay would require pipelining the output of the multiplexer using a flip-flop and asserting the select signals one clock cycle in advance. This method uses more macrocells than method B, but uses fewer pins.

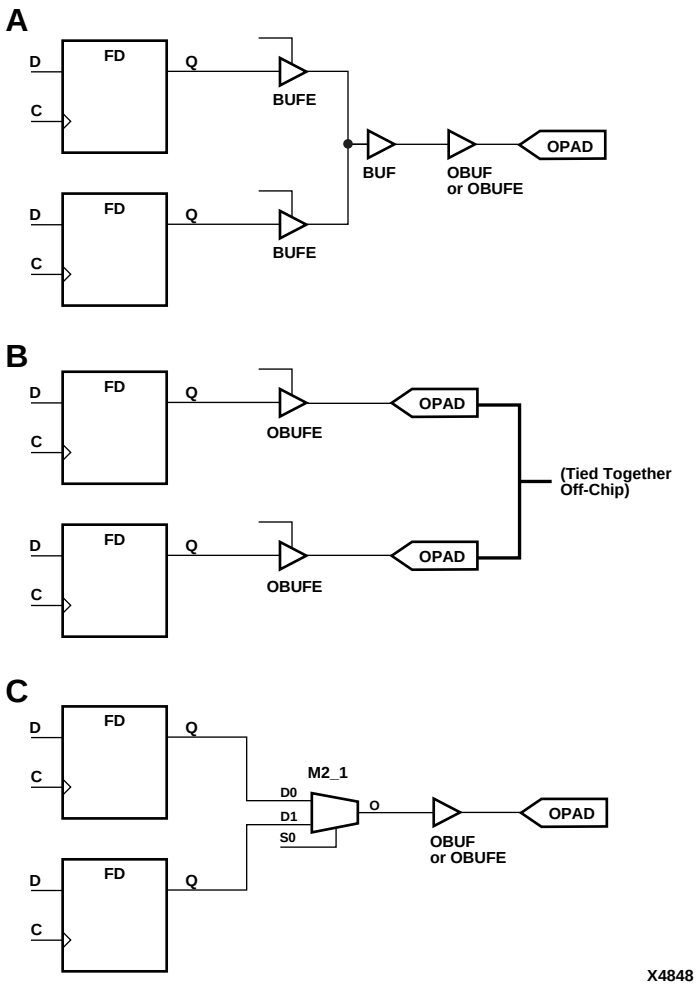

Figure 4-3 Methods of Multiplexing Tristate Signals

Combinatorial 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.

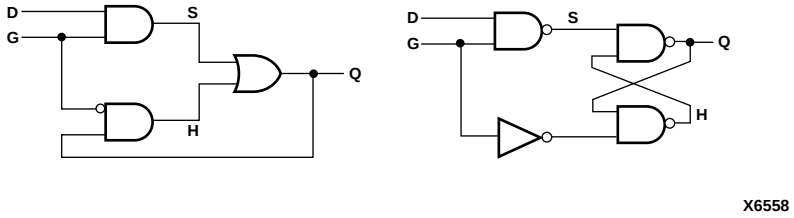

Figure 4-4 Simple Mux and Cross-Coupled-NAND Latches

A timing malfunction can occur if the logic is divided between two separate macrocells by the fitter. The “Malfunction of Physical Implementation” figure illustrates what can happen.

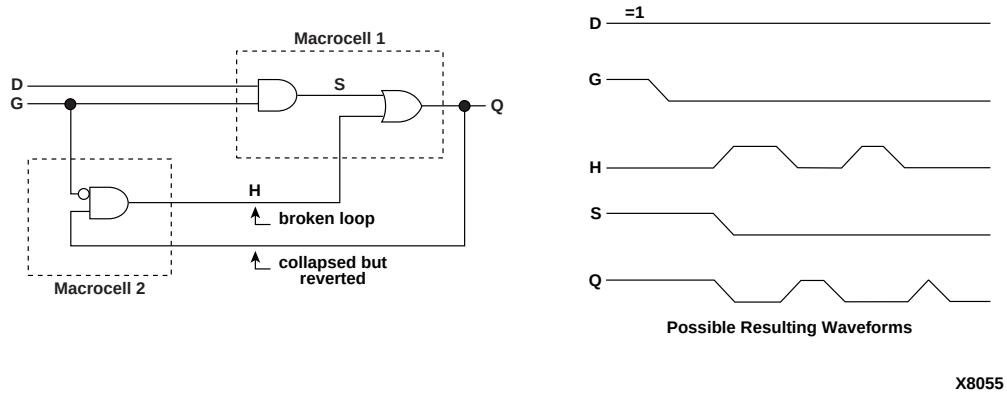

Figure 4-5 Malfunction of Physical Implementation

If you implement the D-type latches with proper redundant logic, the problem does not occur. The “D-type Latch Solutions” figure shows two solutions for schematic implementation of D-type latches.

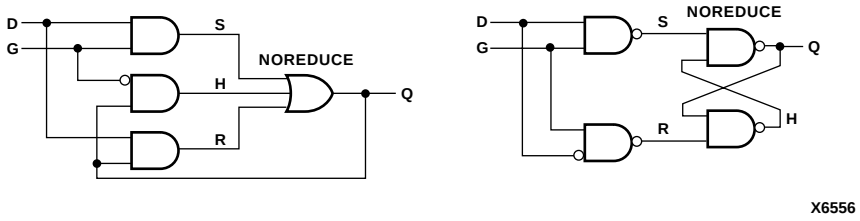

Figure 4-6 D-type Latch Solutions

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

Appendix A

Attributes

Introduction

This appendix describes all of the attributes that you can place into your schematic for an XC9500, XC9500XL or XC9500XV design. The attributes supported are as follows:

- BUFG=CLK | OE | SR
- COLLAPSE
- FAST
- FILE=*filename*
- INIT= R | S
- KEEP
- LOC=*pin_name* | *FBnn*
- NOREDUCE
- PERIOD=*timespec*
- PROHIBIT=P*nn*, P*nn* ...
- PWR_MODE=LOW | STD
- PART=*part_type*
- SLOW
- TNM=*time_group*
- TS*nn*=*time_spec*

Except where noted, attributes can be applied to either component instances or their output nets.

Inputs to Global Nets — BUFG

Applicable Elements

Input buffers (IBUF)

Description

Maps the tagged input buffers to a global net.

BUFG=CLK applied to an IBUF is equivalent to using a BUFG symbol. BUFG=OE applied to an IBUF is equivalent to using a BUFGTS symbol. BUFG=SR applied to an IBUF is equivalent to using a BUFGSR symbol.

Syntax

```
BUFG={CLK | OE | SR }
```

where CLK, OE, and SR indicate clock, output enable, or set/reset, respectively.

Schematic

Attached to an IBUF instance or the input pad net connected to an IBUF input.

UCF/NCF file

Assign to an IBUF instance or the input pad net connected to an IBUF input. This statement maps the signal named “clk1” to a global clock net.

```
NET clk1 BUFG=CLK ;
```

Collapsing a Node — COLLAPSE

Applicable Elements

Internal combinational logic nodes.

Description

Forces a node to be collapsed into all of its fanouts

Syntax

COLLAPSE

Schematic

Attached to a logic symbol or its output net.

UCF/NCF file

This statement forces the logic driving net ABC to collapse into all its fanouts.

```
NET ABC COLLAPSE ;
```

Target Device Selection — PART

Applicable Elements

The design

Description

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 device names supported by the software.

Syntax

The format of the PART value is as follows:

PART=*dddd-ss-pppp*

dddd is the device number with optional “XC” prefix, for example 95108 or XC95108, 9536XL or XC9536XL

ss is the speed grade, for example 10

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

Note: You must specify a complete part value in the PART attribute; you may not use wildcard symbols (*).

Schematic

Instantiate a CONFIG symbol and attach the PART attribute to it.

Clock Cycle Time — PERIOD

Applicable Elements

Clock nets.

Description

A convenient way of defining a clock period for registers attached to a particular clock net

Syntax

PERIOD=*period* [*units*]

where

period is the required clock period

units is an optional field to indicate the units for the clock period. The default is nanoseconds, but the timing number can be followed by ps, ns, us, or ms to indicate the intended units.

Schematic

Attached to a clock input pad net.

```
PERIOD=40
```

UCF/NCF file

This statement assigns a clock period of 40 ns to the input pad net named ABC.

```
NET ABC PERIOD=40 ;
```

Constraints Editor

Period timing constraints can be entered in the global tab for each input pad signal used as a clock.

Behavioral Modules — FILE

Applicable Elements

Custom behavioral module symbols.

Description

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 do not need a FILE attribute.

Specify filename either with or without extension. The software will search for acceptable netlist or equation (Plusasm) files. Specify the directory path if necessary.

Syntax

FILE=[*path*]*filename*[*.extension*]

Where *extension* is one of .edn, .edf, .edif, .sedif, .xnd, and .pld.

Schematic

Attach to a custom behavioral module symbol.

Pin and Function Block Assignment — LOC

Applicable Elements

I/O pads or internal logic components.

Description

Use the LOC=*pin_name* attribute on a PAD symbol or the connected pad net to assign the signal to a specific device pin. The PAD symbols are IPAD, OPAD, IOPAD, and UPAD.

Use the LOC=FB*nn* [*_mm*] attribute on any internal logic symbol, its output net, any output or I/O pad symbol or its connected pad net to assign the logic function to a specific function block or macrocell in the target device.

Syntax

The pin name is P*nn* for PC, PQ or VQ packages; the *nn* is a pin number. The pin name is *rc* (row, column) for BG or PG packages. Examples are LOC=P24 and LOC=G2. For function block assignment,

use LOC=FB*nn* on an internal logic component or its output net. To assign a logic element to a specific macrocell, use LOC=FB*nn_mm*.

LOC=P*nn*

LOC=*rc*

LOC=FB*nn*

LOC=FB*nn_mm*

Note: Pin assignment using the LOC attribute is not supported for bus pads such as OPAD8.

Schematic

Attach LOC=P*nn* | *rc* to a pad symbol or attached pad net. Attach LOC=FB*nn*[*_mm*] to an internal symbol, its output net, an output pad symbol or its attached pad net.

UCF/NCF File

```
NET ABC LOC=P12;
```

Contraints Editor

Location constraints for input and output pad signals can be entered in the Ports tab.

Pin Reservation — PROHIBIT

Applicable Elements

Unused device pins.

Description

PROHIBIT allows you to reserve device pins for later use, or simply to prevent them from being used at all. For instance, if you anticipate design changes in the future and want to set traces on your printed circuit board now, you can use PROHIBIT to prevent the fitter from using pins associated with those traces. Then, when you decide to use the traces, you can use the LOC attribute to assign those pins to new input/output buffers you place in your design.

Syntax

The pin name is P*nn* for PC, PQ or VQ packages; the *nn* is a pin number. The pin name is *rc* (row, column) for BG or PG packages. The PROHIBIT attribute can also accept a comma-separated list of pin names. Examples are PROHIBIT=P24 and PROHIBIT=G2.

PROHIBIT=P*nn*[P*nn*]...

Note: The syntax PROHIBIT=Pnn:Pmm will not work since CPLD has pin names like PA20. This prevents range definition.

Schematic

Instantiate a CONFIG symbol and attach the PROHIBIT attribute to it.

UCF/NCF File

```
PROHIBIT=P12,P13,P14;
```

Constraints Editor

PROHIBIT constraints can be entered using a dialog box provided in the Ports tab.

Power Setting — PWR_MODE

Applicable Elements

Internal logic components.

Description

By default, all macrocells operate in the standard power mode, providing the fastest possible speed. You can change the default to low-power using the Design Manager option. To set the power mode on a specific logic function in your design, apply the PWR_MODE attribute to the symbol or its output net.

Syntax

```
PWR_MODE=LOW|STD
```

Schematic

Attached to an internal symbol or its output net.

UCF/NCF

```
INST ABC PWR_MODE=LOW;
```

Preserving a Node — KEEP

Applicable Elements

Internal combinational nodes.

Description

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

The KEEP attribute inhibits collapsing of a logic function into any of its fanouts.

Syntax

KEEP

Schematic

Attach to an internal symbol or its output net.

UCF/NCF

```
NET ABC KEEP;
```

Register Preload State — INIT

Applicable Elements

Registers and registered macros.

Description

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). This attribute can be applied to flip-flops or any component containing a register, or their output nets.

Syntax

INIT=R|S

Schematic

Attach to a flip-flop or any component containing a register, or its output net.

UCF

The following sets a preload value of 0 on element ABC..

```
INST ABC INIT=R;
```

Output Slew Rate — FAST, SLOW

Applicable Elements

Output and I/O pads.

Description

The FAST attribute can be placed on an output pad or I/O pad to select the fast slew-rate operation of the CPLD output-pin driver.

The SLOW attribute selects the slew-rate limited control.

Syntax

FAST

or

SLOW

Schematic

Attached to an OPAD or IOPAD instance or the connected pad net.

UCF/NCF file

This statement establishes a slow slew rate for an instantiation of output signal ABC.

```
NET ABC SLOW ;
```

Constraint Editor

FAST and SLOW slew-rate can be selected for any output pad signals in the Ports tab.

Minimization of Redundant Logic — NOREDUCE

Applicable Elements

Internal combinational nodes.

Description

The NOREDUCE attribute tells the fitter to disable Boolean logic minimization for the attached component. You need to use the NOREDUCE attribute if you want to specify redundant logic in a portion of your design to avoid a potential race condition. The NOREDUCE attribute also identifies the output node of a combinational feedback loop. For example, you would use NOREDUCE on the output gate when designing combinational feedback latches.

Syntax

NOREDUCE

Schematic

Attached to a logic symbol or its output net.

UCF/NCF file

To prevent boolean reduction at node ABC:

```
NET ABC NOREDUCE ;
```

Timing Specifications — TS*identifier*

Applicable Elements

Timing paths between I/O pads and flip-flops.

Description

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, and the underscore character (_). The value of the T-spec attribute consists of a FROM-TO expression specifying the timing requirements between specific end points.

Syntax

The full syntax is shown as follows:

TS*identifier***=FROM:***source_group***:TO:***dest_group***:***delay***[***units***]**

or

TS*idr***=PERIOD:***clock_group***:***delay***[***units***]**

The parameters *source_group* and *dest_group* 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 parameter *clock_group* must be a TNM identifier placed on a clock input pad.

The *delay* parameter defines the maximum delay for the attribute, using nanoseconds as the default unit of measurement. The default units are nanoseconds, but the timing number can be followed by ps, ns, us, ms, GHz, MHz, or kHz to indicate the intended units.

Schematic

Instantiate a TIMESPEC symbol and attach one or more T-spec attributes to it.

UCF/NCF

```
TIMESPEC TSnn=time_spec;
```

Constraints Editor

Clock period timing constraints can be entered in the Global tab. Input setup time and clock-to-output delay can be entered for specific pads in the Ports tab, or for all pads related to a given clock in the global tab. Combinational pad-to-pad delays can be entered in the Advanced tab, or for all pad-to-pad paths in the Global tab.

Timing Group Name — TNM

Applicable Elements

I/O pads and flip-flops.

Description

The TNM attribute creates a timing group name for all pads or flip-flops to which it applies. These timing group names can tehn be used in timing constraints (T-specs) to define groups of timing path end-points.

Syntax

TNM=*group_name*

Schematic

Attach to pad instances, pad nets or flip-flops. Can also attach to internal nets; the software will forward-trace the TNM to all connected flip-flops or output pads.

UCF/NCF

```
INST FLOP1 TNM=ABC;
```

Constraints Editor

Timing group names can be assigned to pad and flip-flop elements in the Advanced tab.

Timing Group Definitions

Applicable Elements

I/O pads and flip-flops.

Description

Timing group definitions create new timing group names based on combinations and/or filters of existing timing group names.

Syntax

group_name=group1[*:group2*] *...*

Schematic

Instantiate a TIMEGRP symbol and attach one or more timing group definitions to it.

UCF/NCF

TIMEGRP *group_name*=...;

Constraints Editor

New timing groups can be created in the Advanced tab.

Appendix B

CPLD Library Selection Guide

This appendix contains two tables listing CPLD components and LogiBLOX modules respectively.

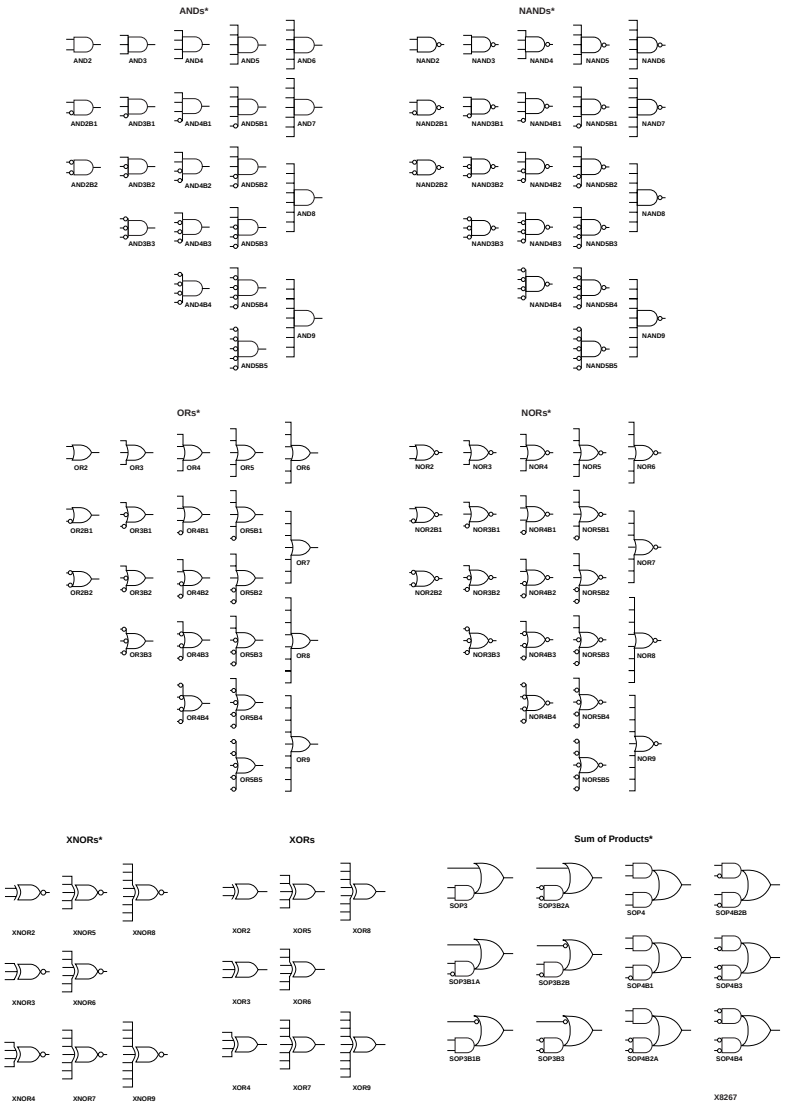

Figure B-1 CPLD Schematic Components

Table B-1 CPLD Components

Component Name	Description/Features
Buffers and Inverters	
BUF*, BUF4, BUF8, BUF16	Non-inverting buffer
BUFE*, BUFE4, BUFE8, BUFE16	Internal tristate buffer with active-high enable. Not available on XC9500XL or XC9500XV devices.
BUFG*	Global clock input buffer
BUFGSR* BUFGTS*	Global asynchronous set/reset input buffer Global tristate control input buffer
BUFT*, BUFT4, BUFT8, BUFT16	Internal tristate buffer with active-low enable. Not available on XC9500XL or XC9500XV devices.
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

Table B-1 CPLD Components

Component Name	Description/Features
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

Table B-1 CPLD Components

Component Name	Description/Features
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
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

Table B-1 CPLD Components

Component Name	Description/Features
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	Laudable cascabel 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

Table B-1 CPLD Components

Component Name	Description/Features
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
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

Table B-1 CPLD Components

Component Name	Description/Features
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

Table B-1 CPLD Components

Component Name	Description/Features
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
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	Tristate output buffer with active-high enable
OBUFT*, OBUFT4, OBUFT8, OBUFT16	Tristate output buffer with active-low enable
OPAD*, OPAD4, OPAD8, OPAD16	Output pad

Table B-1 CPLD Components

Component Name	Description/Features
Miscellaneous	
GND*	Ground-connection signal tag
VCC*	VCC-connection signal tag
TIMEGRP*	Timing specification group table
TIMESPEC*	Timing requirement specification table
CONFIG*	Used to carry PART and PROHIBIT attributes

* Primitive symbols (all others are macros)

Table B-2 LogiBLOX Modules

Module	Description
ACCUMULATOR	Adds data to or subtracts it from the current value stored in the accumulator register.
ADDER/ SUBTRACTER	Adds or subtracts two data inputs and a Carry input.
CLOCK DIVIDER	Generates a clock pulse whose period is a multiple of the clock input period.
COMPARATOR	Compares the magnitude or equality of two values.
CONSTANT	Forces a constant value onto a bus.
COUNTER	Generates a sequence of count values.
DATA REGISTER	Captures the input data on active Clock transitions.
DECODER	Routes input data to 1-of-n lines on the output port
INPUT/OUTPUT	Connects internal and external pin signals
MULTIPLEXER	Type 1, Type 2 — Routes input data on 1-of-n lines to the output port.
PAD	Simulates an input/output pad.
SHIFT REGISTER	Shifts the input data to the left or right.

Table B-2 LogiBLOX Modules

Module	Description
SIMPLE GATES	Type 1, Type 2, Type 3 — Implements the AND, INVERT, NAND, NOR, OR, XNOR, and XOR logic functions.
TRISTATE BUFFER	Creates a tri-stated internal data bus. Not available on XC9500XL or XC9500XV devices.

Appendix C

Fitter Command and Option Summary

This appendix describes how to invoke the CPLD fitter, and the commands used to prepare functional and timing simulation models. All of the available fitter options are described. This chapter contains the following sections:

- "Design Manager"
- "CPLD Command"

Design Manager

The Design Manager invokes the Flow Engine (fitter) and option templates to control the fitting of your design.

Invoking the Fitter

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. Select the target device. If your schematic contains a PART attribute, the specified part appears in the **Implement** dialog box. You can override any of the fields in the **Part Selector** dialog. Otherwise, select either XC9500, XC9500XL, or XC9500XV as the family. By default, package and speed are automatically

selected by the fitter. You can select a specific device, package or speed in any of the fields.

3. Open the Constraints Editor to enter timing and pad location constraints (optional). Select **Utilities** → **Constraints Editor**
4. Select options for design implementation. Select **Design** → **Implement**
5. The Design Implementation Option menu appears. Select either the **Optimize Speed** (default) or **Optimize Density** template.
6. To adjust specific fitter options, select **Edit Template**. Then select from the tabs all the options you want to use and press **OK**. See *Fitter Command Parameters* later in this chapter.
7. To run the fitter, click once on the **run** key found in the Flow Engine.

Fitter Options

This section describes fitter parameters that can be entered from the Design Manager or when using the command line on a workstation.

The Implementation Options menu contains five tabs of options for the fitter. The following summarizes fitter options:

- **Basic** → **Default Output Slew Rate** — sets default output slew-rate to **FAST** or **SLOW** (default is **FAST**).
- **Basic** → **Macrocell Power Setting** — Sets default power mode for all macrocells in the design to standard or low-power (default is **Std** power).
- **Basic** → **Create Programmable Ground Pins** — creates additional ground pins on unused I/Os (default is **OFF**).
- **Basic** → **Use Design Location Constraints**— if this is not checked, the program temporarily ignores all LOC attributes in the design, allowing the fitter to assign the locations of all I/O pins (default is **ON**).
- **Basic** → **Use Timing Constraints** — turn this selection off if you want to temporarily ignore all timing specification attributes in the design (default is **ON**).

- **Basic** → **Use Global Clock(s)** — Select this option to automatically use global clocks (**GCK**) for ordinary input signals used as clocks. The global clock may allow you to meet your timing constraints more easily. By default, this option is **ON**.
- **Basic** → **Use Global Output Enable(s)** — Select this option to automatically use global output enable (GTS) for ordinary input signals used as output enable constraints. Global output enable may allow you to meet your timing constraints more easily. By default, this option is **ON**.
- **Basic** → **Use Global Set/Reset** — Select this option to automatically use global set / reset (GSR) for ordinary input signals used as asynchronous clear or preset. By default, this option is **ON**.
- **Advanced** → **Collapsing Input Limit** — The maximum number of function block inputs allowed as a result of logic collapsing. Default is 36.
- **Advanced** → **Collapsing Pterm Limit** — The maximum number of product terms allowed as a result of collapsing (default=20 on **Optimize Speed** template; 90 on **Optimize Density** template).
- **Advanced** → **Use Multilevel Logic Optimization** — Spends additional time transforming the logic in your design to new logical structures that achieve better performance and density (default=**ON**).
- **Advanced** → **Use Timing Optimization** — enables the global timing optimization performed by the fitter; if this option is not selected, only paths with T-specs specified in the design are optimized to improve timing (default is **ON** in **Optimize Speed** template, **OFF** in **Optimize Density** template).
- **Advanced** → **Enable D to T-Type Transform Optimization** — if this box is checked (default), the fitter transforms between D-type and T-type registers.
- **Advanced** → **Use Advanced Fitting** — Select this option to enable an advanced fitting strategy that favors placing signals with common inputs in the same function block. This usually allows you to pack more logic into the same device. Disable this option if the software has trouble fitting a design that used to fit with an older version of software (by default, this option is **ON**).

This option applies to XC9500 devices only (no XC9500XL or XC9500XV devices).

- **Advanced** → **Use Local Macrocell Feedback** — enables the software to use local feedback in XC9500 devices (except XC9536) whenever possible. The local feedback path takes less time than the global feedback path. Using local feedback can speed up your design but can make it difficult to keep the same timing after a design change (default is **OFF**).
- **Advanced** → **Use Local Pin Feedback** — enables the software to use local I/O pin feedback in XC9500 devices whenever possible. The software uses the pin feedback path instead of the FastCONNECT path for output pin signals that do not have 3-state control or slow slew rate (by default, this option is **OFF**).
- **Interface** → **Macro Search Path** — Use this option to add the specified search path to the list of directories to search when resolving file references (that is, files specified in the schematic with FILE-*filename* property). This option also supplies paths for macros (*design_name*.nmc) or other directories containing NGO files. Specify a macro search path or click **Browse** to look for a path to add as a macro search path. To specify multiple search paths, type in each directory name separated by a colon (:). A semicolon is automatically appended when you use the **Browse** button to select multiple search paths.
- **Interface** → **Rules File** — Use this option only to specify a custom rule file used for translating netlist formats not normally supported by the software.
- **Interface** → **Create I/O Pads from Ports**— Creates I/O pads from ports. Select this option only if your netlist does not contain either PAD symbols or top-level ports defining external signals (you do not normally need to use this option). By default, this option is **OFF**.
- **Timing Reports** → **Produce Post Layout Timing Report** — generates static timing report.
- **Timing Reports** → **Timing Report Format** — Select **Summary** to generate a report that contains summary information and design statistics. Select **Detailed** to generate a report that lists delay information for all nets and paths.

- **`Programming`** → **`Signature/User Code`** — Enter a unique text string in this field to identify the signature data. You can enter a string of up to four alphanumeric characters. The device programmer can read the signature, and the person running the device programmer can verify that the correct configuration data file is loaded. Use the JTAG Programmer to identify the configuration data signature (usercode) of a programmed XC9500 device.
- **`Programming`** → **`Jedec Test Vector File`** — Use this option to select a test vector file (.tmv) produced by the XABEL compiler to incorporate into the JEDEC programming file to a perform functional test (INTEST).

CPLD Command

The **`cpld`** command invokes the CPLD design implementation software (the fitter). The command is run in a UNIX command window and is only supported on UNIX workstations. Your current working directory must be set to the project directory which contains your design source netlist files before invoking cpld.

Invoking the Fitter

The format of the cpld command is:

`cpld` [*options*] *design_name*

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

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

Schematics must first be translated into either an EDIF-formatted netlist (*design_name*.edif). XNF formatted netlists are also acceptable from tools that do not have EDIF, but EDIF is preferred. Xilinx strongly recommends that you generate new EDIF netlists from existing schematics rather than trying to reuse existing XNF netlists. Also, your netlists must be created using schematic capture libraries provided by Xilinx or your CAE tool vendor for use with the current version of Xilinx software.

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

1. Synopsys Design Compiler or FPGA Compiler netlist (*design_name*.sxnf)
2. Xilinx PLUSASM equation file (*design_name*.pld)
3. XNF netlist (*design_name*.xnf)
4. Synopsys Design / FPGA Compiler EDIF netlist (*design_name*.sedif)
5. EDIF netlist (*design_name*.edn, *design_name*.edf or *design_name*.edif)
6. Xilinx NGO (unexpanded) database file (*design_name*.ngo)
7. Xilinx NGD (expanded) database file (*design_name*.ngd)

Fitter Options

The *[options]* field of the cpld command represents an optional list of one or more command-line parameters. Invoking the **`cpld`** 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 **`cpld`** command-line parameters that apply to schematic design entry:

- **`-autoslewpwr`** — reduces slew rate before reducing power mode if **`autopwrslew`** is enabled.
- **`-autopwrslew`** — reduces power mode and / or slew rate if timespecs can still be met or if no timespecs apply.
- **`-detail`** — produces a detailed path timing report (*design_name*.tim) instead of the default summary report.
- **`-grounds`** — creates programmable ground pins on unused I / Os.
- **`-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.
- **`-inputs`** <*n*> — maximum number of function block inputs allowed as a result of logic collapsing. Default is 36.

- **`-localfbk`** — uses local feedback. Enables the software to use local feedback whenever possible. The local feedback path takes less time than the global feedback path. Using local feedback can speed up your design but can make it difficult to keep the same timing after a design change. XC9500 only.
- **`-loweffort`** — low fitting effort, to save processing time.
- **`-lowpwr`** — uses the low-power mode by default for all macrocells in the design (default is normally standard power).
- **`-nodt`** — disables transformation between D-type and T-type registers.
- **`-nogck`** — disables global clock optimization.
- **`-nogsr`** — disables global set/reset optimization
- **`-nogts`** — disables global output-enable (GTS) optimization.
- **`-nomlopt`** — disables multi-level logic optimization.
- **`-nota`** — do not generate a summary static timing report.
- **`-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.
- **`-notsim`** — disables generation of timing simulation file (.nga).
- **`-nouim`** — disables implementation of AND functions in FastConnect. XC9500 only.
- **`-noxor`** — disables transformation of sum-of-product XOR logic into macrocell XOR gates.
- **`-p`** ***part_type*** — specifies the target device type or set of devices from which to choose (default is automatic device selection from the XC9500 family); where *part_type* can be:
  - 9500 = any XC9500 family device (auto selection)
  - 9500xl = any XC9500XL family device (auto selection)
  - 9500xv= any XC9500XV family device (auto selection)
  - "*95ddd[xl][-ss][-pppp]*" — where *95ddd* 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).

- **`-pinfbk`** — uses pin feedback. Enables pin feedback whenever possible. The software uses the pin feedback path instead of the FastCONNECT path for output pin signals that do not have 3-state control or slow slew rate. XC9500 only.
- **`-pinlock`** — 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* — the maximum number of product terms allowed as a result of collapsing (default=20).
- **`-s`** *signature* — specifies the user signature string (up to 4 alphanumeric characters) to be programmed into the device for identification purposes (default is the design name).
- **`-slowslew`** — applies slow output slew-rate as default (default is fast).
- **`-ucf`** — reads user constraints from *filename*.ucf. By default, *design_name*.ucf is read if it exists.
- **`-xactfit`** — Use this option only if you have a design implemented in XACT v6 and cannot get the same pinout using the current software. The default is advanced fitting.

Appendix D

Simulation Summary

This appendix contains the following sections on simulation:

- “Timing Simulation”
- “Simulation from Design Manager”
- “Simulation on Workstation Command LIne”
- “Simulating Power-On Initialization”

Timing Simulation

The Design Manager / Flow Engine can produce timing simulation data for use in a third party simulation tool.

This section describes how to prepare a simulation model file for functional and timing simulation in the Design Manager and Workstation command line environments.

Simulation from Design Manager

The Design Manager produces timing simulation data automatically when you run the fitter. To produce timing simulation go to the **`Setup Options`** template and check the box labelled **`Produce Timing Simulation Data`**. A timing simulation netlist is automatically generated when the Flow Engine runs.

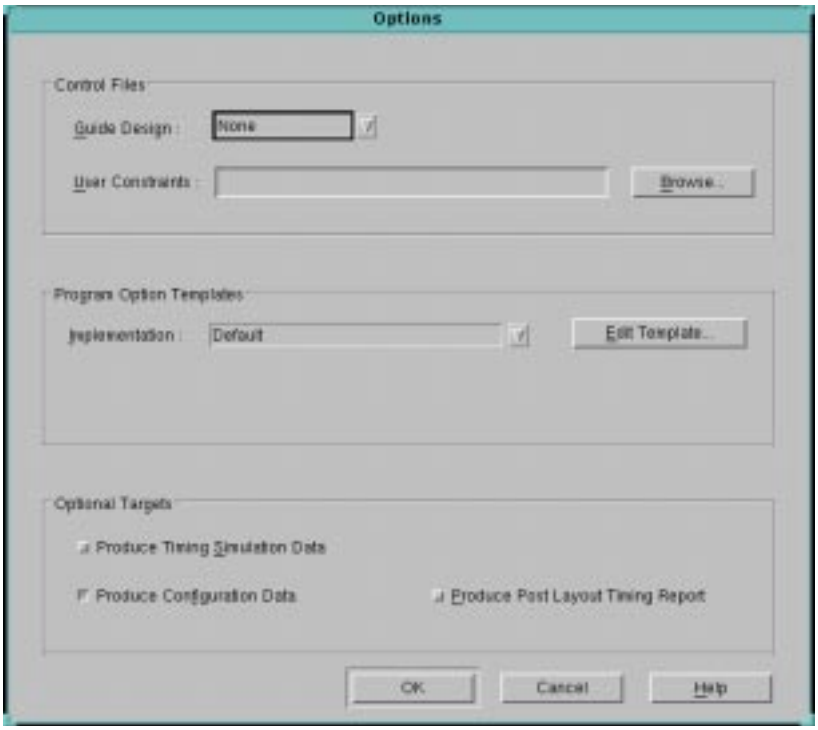

Figure D-1 Options

By default the simulation data is produced in EDIF format. Format is set from the **Implementation Options** dialog box; if you want to select another format, go to the **Interface** tab and click the down arrow adjacent to **Format,** then select from the supported formats.

When you implement the design, the Flow Engine produces timing simulation data files. Each time the data is produced, it is automatically exported to your design directory.

You can now use these files to simulate the design with a supported third party simulation tool.

Simulation on Workstation Command LIne

The commands **ngd2vhdl** and **ngd2ver** and **ngd2edif** give you the ability to simulate vhdl, verilog, and edif designs on a workstation command line. See the *Development System Reference Guide* for instructions on using these three programs.

NGD2EDIF

The NGD2EDIF program produces an EDIF 2.0.0 netlist in terms of the Xilinx primitive set, allowing you to simulate pre- and post-route designs.

Syntax

To invoke the NGD2EDIF translation program from the UNIX or DOS command line, enter the following:

ngd2edif [*options***]** *infile***[.ngd|.nga] [***outfile***[.edn]]**

where:

Options can be any number of the NGD2EDIF options listed in this section. They do not need to be listed in any particular order. Separate multiple options with spaces.

*infile***[.ngd** | **.nga]** indicates the input file. If you enter a file name with no extension, NGD2EDIF looks for a file with an .nga extension and the name you specified. If you want to translate an NGD file, you must enter the .ngd extension. Without the .ngd extension NGD2EDIF does not use the NGD file as input, even if there is no NGA file present.

*outfile***[.edn]** is the name of NGD2EDIF's output file if you want to name it other than the root NGD design name. If you do not give an extension, .edn is added.

If you are using the Viewlogic design entry tools, it is important that the *outfile* name be different from the original design name, to avoid conflict with the original WIR and EDIF files.

Options

- –a (Write All Properties)

  The –a option causes NGD2EDIF to write all properties into the output EDIF netlist. The default is to write only timing delay properties and certain other properties that define the behavior of the design logic. In most cases the –a option is not necessary; your simulation vendor can tell you if it is required for the vendor's flow.

- –n (Generate Flattened Netlist)

The –n option writes out a flattened netlist.

- –v (Vendor)

  –v *vendor*

  The –v option specifies the CAE vendor toolset that uses the resulting EDIF file. Allowable entries are **viewlog** (for Viewlogic) and **mentor**.

  The –v option customizes the output EDIF file for the specified vendor's simulator.

- –w (Overwrite Output)

  The –w option indicates to overwrite the output file.

NGD2VHDL

The NGD2VHDL program translates your design into a VITAL 95 IEEE compliant VHDL file containing a netlist description of the design in terms of Xilinx simulation primitives. The VHDL file can be used to perform a back-end simulation by a VHDL simulator.

Syntax

The following syntax translates your design to a VHDL file:

ngd2vhdl [*options***]** *infile***.[ngd|.nga] [***outfile***[.vhd]]**

where:

Options can be any number of the NGD2VHDL options listed in this section. They do not need to be listed in any particular order. Separate multiple options with spaces.

Infile [**.ngd** | **.nga**] is the input NGD or NGA file. If you enter a file name with no extension, NGD2VHDL looks for a file with an .nga extension and the name you specified. If you want to translate an NGD file, you must enter the .ngd extension. Without the .ngd extension NGD2VHDL does not use the NGD file as input, even if there is no NGA file present.

Outfile[**.vhd**] indicates the file to which the VHDL output of NGD2VHDL is written. Default is *infile*.vhd (*infile* is the same root name as the input file). The SDF file has the same root name as the VHDL file.

Options

- –gp (Bring Out Global Reset Net as Port)

  The –gp option causes NGD2VHDL to bring out the global power-on simulation signal (which is connected to all flip-flops and latches in the physical design) as a port on the top-level entity in the output VHDL file name PRLD.

  If you do not specify the -gp option, the output.vhd file will contain an ROC cell which automatically pulses the internal PRLD net at the beginning of the simulation sequence.

- –tb (Generate Testbench File)

  The –tb option writes out a testbench file. The file has a .tvhd extension.

  The default top-level instance name within the testbench file is UUT. If you enter a –ti (Top Instance Name) option, the top-level instance name is the name specified by the –ti option.

- –w (Overwrite Existing Files)

  The –w option causes NGD2VHDL to overwrite the output files if they already exist. By default (no –w specified) NGD2VHDL does not overwrite existing files.

NGD2VER

The NGD2VER program translates your design into a Verilog HDL file containing a netlist description of the design in terms of Xilinx simulation primitives. The Verilog file can be used to perform a back-end simulation by a Verilog simulator.

Syntax

The following syntax translates your design to a Verilog file:

ngd2ver [*options***]** *infile***[.ngd|.nga] [***outfile***[.v]]**

Options can be any number of the NGD2VER options listed in this section. They do not need to be listed in any particular order. Separate multiple options with spaces.

Infile [**.ngd** | **.nga**] is the input NGD or NGA file. If you enter a file name with no extension, NGD2VER looks for a file with an .nga extension and the name you specified. If you want to translate an

NGD file, you must enter the .ngd extension. Without the .ngd extension NGD2VER does not use the NGD file as input, even if there is no NGA file present.

Outfile[**.v**] indicates the file to which the Verilog output of NGD2VER is written. Default is *infile*.v (*infile* is the same root name as the input file). The SDF file has the same root name as the Verilog file.

Options

- –gp (Bring Out Global Reset Net as Port)

  The –gp option causes NGD2VER to bring out the global power-on simulation signal (which is connected to all flip-flops and latches in the physical design) as a port on the top-level module in the output Verilog file named PRLD.

- –tf (Generate Test Fixture File)

  The –tf option generates a test fixture file. The file has a .tv extension, and it is a ready-to-use template test fixture Verilog file based on the input NGD or NGA file.

  If you are using a Cadence Verilog simulator, you can run the simulator by entering

  `verilog` *design*`.tv` *design*`.v`,

  using the output V and TV files from NGD2VER. You can then add more design-specific stimuli to this file to fit your needs.

- –w (Overwrite Existing Files)

  The –w option causes NGD2VER to overwrite the output files if they already exist. By default (no –w specified) NGD2VER does not overwrite existing files.

Simulating Power-On Initialization

The XC9000 component library for all tools directly supported by Xilinx and most third-party tools contain functional simulation models for all of the primitive symbols. Models for registered components contain a global net named PRLD that will reset the registers to zero when pulsed high at the beginning of functional simulation.

In the timing simulation netlists produced by the software, a net named PRLD is added to the design to represent the device power-on

condition. When this PRLD net is pulsed high, all registers in the device are initialized to the states specified by INIT attributes in your design.

To simulate the XC9000 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 values to prevent registers from lapsing back into the "unknown" state. You should hold your device inputs at valid logic values long enough to propagate to all the registers before returning PRLD low.

In EDIF simulation netlists, PRLD is a global internal signal. If you are simulating based on an EDIF output file (**.edn**), you must drive the PRLD signal low throughout the remainder of your simulation. Otherwise the registers in the design will remain stuck in the unknown state.

If you create a VHDL or Verilog HDL timing simulation netlist, it also contains a PRLD net used to initialize all the registers in the design. The PRLD net is driven by either a pullup resistor, for Verilog HDL netlists, or by a Reset-on-Configuration (ROC) pulse generator for VHDL netlists, unless you bring it out as a port by specifying the **-gp** option on the **ngd2vhdl** or **ngd2ver** command line.