ISE 4 User Guide

About This Manual

This chapter contains the following sections:

About the ISE 4 User Guide

The Integrated Synthesis Environment (ISE) from Xilinx is an integrated tool suite that enables you to produce, test, and implement designs for Xilinx FPGAs or CPLDs. The tools cover all aspects of the design flow, from design entry to bitstream generation and downloading. You can directly access the Internet from many of the ISE applications for user support and tool updates.

This ISE 4 User Guide:

- Provides an overview of ISE 4.x.
- Describes the ISE 4.x design environment.
- Explains how to create a project.
- Summarizes each of the steps in the design flow, including design entry, constraint entry, synthesis, simulation, implementation, and device programming.
- Briefly describes each tool in the ISE 4.x suite.
- Provides references to additional help, documentation, and support.

Manual Contents

- Chapter 1, "Introduction," describes ISE in general, lists supported platforms and architectures, and describes partner tools available for use with ISE. It also includes references to sources of additional assistance, including tutorials, books, online help, and technical support.
- Chapter 2, "Project Navigator,"describes Project Navigator, the primary user interface for ISE. Project Navigator integrates the design entry, constraint entry, implementation, synthesis, simulation, and device programming tools and processes that facilitate design production.
- Chapter 3, "Projects," describes projects and project management generally, and describes the specific ISE project creation process, which includes specifying a directory for the project, identifying the Xilinx device you want to target for your design, choosing a project flow, and adding and creating source files.
- Chapter 4, "Design Flow," describes the overall design process, including Design Entry, Constraint Entry, Synthesis, Simulation, Implementation, and Device Programming.
- Chapter 5, "HDL," describes HDL design sources (VHDL, Verilog, and ABEL-HDL), the HDL Editor (a general purpose text editor that is HDL language sensitive), and the HDL library mapping feature.
- Chapter 6, "State Diagrams," describes the integration of the supported state diagram entry tools with ISE. For state machine design entry, ISE includes integrated support for StateCAD® and StateBench™.
- Chapter 7, "Schematic Sources,"explains how to use schematic sources in ISE projects. It also contains an overview of the basic concepts for using the Engineering Capture System (ECS).
- Chapter 8, "LogiBLOX," describes LogiBLOX—a design tool for creating high-level modules such as counters, shift registers, and multiplexers for XC4000 / XL / XLA, Spartan / XL, and 9500 / XL / XV designs.
- Chapter 9, "CORE Generator," describes the Xilinx CORE Generator System, a design tool that delivers parameterizeable COREs optimized for Xilinx FPGAs.

- Chapter 10, "Implementation," explains the implementation design process that converts the logical design represented in the source file into a physical file format that can be implemented in the selected target device.
- Chapter 11 "FPGA Implementation," explains the implementation process for FPGA devices, including FPGA Implementation Flow, FPGA Implementation Reports, FPGA Implementation Options, and FPGA Implementation Tools (Floorplanner, FPGA Editor, and Timing Analyzer).
- Chapter 12, "CPLD Implementation," explains the implementation process for CPLD devices, including CPLD Implementation Flow, CPLD Implementation Reports, CPLD Implementation Options, and CPLD Implementation Tools (Timing Analyzer and ChipViewer).
- Chapter 13, "Device Programming,"explains how to create a bitstream to download to a device, and describes the programming tools PROM File Formatter and iMPACT.

Additional Resources

For additional information, go to http://support.xilinx.com. The following table lists some of the resources you can access from this Web site. You can also directly access these resources using the provided URLs.

Resource	Description and URL
Tutorials	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
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 contains device-specific information on Xilinx device characteristics, including readback, boundary scan, configuration, length count, and debugging http://support.xilinx.com/partinfo/databook.htm

Resource	Description and URL
Xcell Journals	Quarterly journals for Xilinx programmable logic users http://support.xilinx.com/xcell/xcell.htm
Technical Tips	Latest news, design tips, and patch information for the Xilinx design environment http://support.xilinx.com/support/techsup/journals/index.htm

Conventions

This manual uses the following conventions. An example illustrates most conventions.

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.

- Blue text indicates cross-references within a book. Red text indicates cross-references to other books. Click the colored text to jump to the specified cross-reference.
- Blue, underlined text indicates a Web site. Click the link to open the specified Web site. You must have a Web browser and Internet connection to use this feature.

Contents

Chapter 2 Project Navigator

Chapter 3 Projects

Chapter 4 Design Flow

Chapter 5 HDL

Chapter 6 State Diagrams

Chapter 7 Schematic Sources

Chapter 8 LogiBLOX

Chapter 9 CORE Generator

Chapter 10 Implementation

Chapter 11 FPGA Implementation

Chapter 12 CPLD Implementation

Chapter 13 Device Programming

Chapter 1

Introduction

This chapter contains the following sections:

About ISE 4.x

ISE 4.x from Xilinx is a next-generation design environment for programmable logic design. It offers advanced software capabilities that enable designers to efficiently create and verify programmable logic design. Features include:

- HDL Design Entry and Project Management Tools
- HDL Synthesis Engines from Synopsys® and Xilinx
- Incremental Design with Block Level Incremental Synthesis (BLIS)
- Seamless integration with Synplify™ from Synplicity
- Seamless integration with LeonardoSpectrum from Exemplar
- Xilinx Ultra-fast Place and Route
- Customizable Intellectual Property Using CORE Generator

- Graphical State Diagram Entry Using StateCAD
- Seamless Integration with ModelSim™ HDL Simulation Tools
- Automatic Self-checking Testbench Generation Using HDL Bencher

Xilinx Architecture Support

For a list of the most current supported device families and device flows, see the Device Support table in the ISE online help (**Help → ISE Help Contents → Reference → Device Support).** The devices available for your design depend upon the product configuration you purchased, and on the design flow you select for your project.

Installation

For instructions on installing ISE 4.x, see the *ISE Installation Guide and Release Notes.* This document accompanies your software, and is also available on the Xilinx support Web site at http://www.support.xilinx.com/ under Software Manuals. It describes installation procedures, key features, supported devices, and the most critical known issues.

Partner Tools

ISE 4.x supports the following partner tools to provide a complete design environment:

- FPGA Express from Synopsys®

  FPGA Express is integrated into the design flows of ISE 4.x to provide synthesis functions as it has in earlier products.

- ModelSIM™ from Model Technology Incorporated (MTI)

  ModelSIM simulators provide the simulation functions for ISE 4.x.

- LeonardoSpectrum from Exemplar

  LeonardoSpectrum is a synthesis tool that synthesizes VHDL or Verilog and creates an EDIF netlist. An interface into this tool is provided in ISE 4.x. However, in order to use LeonardoSpectrum

with ISE, you must have purchased and installed the program on your computer.

- Synplify and Synplify Pro from Synplicity

  Synplify and Synplify Pro are synthesis tools designed for FPGA synthesis with VHDL, Verilog and mixed languages to create an EDIF netlist. In order to use either tool with ISE, you must separately purchase and install the program on your computer.

Tutorials

Several tutorials are available to assist you in learning ISE 4.x:

- "ISE Tutorial"
- "In-Depth Tutorial"
- "Other Tutorials"

ISE Tutorial

The *ISE Tutorial* is a printed booklet distributed with your ISE software. It describes the creation of a 4-bit counter module. The tutorial includes:

- creating HDL and schematic source files for the design
- functionally simulating the design's logic
- processing the design for device implementation
- using basic timing simulation to test the design in the device

In-Depth Tutorial

An in-depth tutorial, the Watch tutorial, is available on the Xilinx support Web site from http://support.xilinx.com/support/techsup/tutorials/index.htm.

Other Tutorials

The following tutorials are available from the Project Navigator online help:

- CPLD Design Flows
- XPower FPGA Tutorial
- XPower CPLD Tutorial

Online Help

The following online help is available from Project Navigator and its associated tools. To access the help, click the Help menu, or press **F1**.

- "Context Sensitive Help"
- "Project Navigator Help Contents"
- "ISE Help Contents"
- "How to Use Help"
- "Online Documentation"
- "Xilinx on the Web"

Context Sensitive Help

Context sensitive help is available for all ISE 4.x tools. To access context sensitive help, press **F1**.

Project Navigator Help Contents

To access help for Project Navigator as a whole, select **Help** → **Project Navigator Help Contents**.

ISE Help Contents

The ISE Help umbrella lists all help files available from within Project Navigator. To access the ISE help umbrella, select **Help** → **ISE Help Contents**. When the umbrella opens, click a link to access the

indicated help or other support file. The umbrella help menu includes links to the following:

Design Entry

- Project Navigator
- HDL Editor
- StateCAD State Machine Editor
- Schematic Editor (ECS)
- LogiBLOX
- CORE Generator
- Xilinx Constraints Editor

Synthesis

- XST (Xilinx Synthesis Technology)
- FPGA Express
- Synplify
- LeonardoSpectrum

Simulation

- HDL Bencher
- ModelSim Simulator

Implementation

- Floorplanner
- FPGA Editor
- Timing Analyzer
- XPower
- ChipViewer

  Note Chip Viewer help can be accessed from the Chip Viewer window only.

Programming

- iMPACT
- PROM File Formatter

Techniques

- FPGA Design Techniques
- CPLD Design Techniques
- Entering Constraints

Tutorials

- ISE Tutorial
- In-Depth Tutorial
- CPLD Design Flows
- XPower FPGA Tutorial
- XPower CPLD Tutorial

For more information, see the "Tutorials" section above.

Application Notes

- ISE Application Notes

  Note For current versions of ISE Application Notes, see http://www.xilinx.com/apps/appsweb.htm

- FPGA Express Application Notes

Reference

- ISE 4.x Key Features
- Device Support
- Product Licensing
- VHDL Reference Guide
- Verilog Reference Guide
- ABEL Reference Guide

- CPLD Schematic Library
- CPLD Attributes
- Technical Support

How to Use Help

How to Use Help is a Microsoft Windows tutorial which explains how to use an online help file.

Online Documentation

Click the Online Documentation link to open the book collection that accompanies your ISE 4.x software. For more information, see the "Books" section below.

Xilinx on the Web

To access Xilinx services on the World Wide Web, click **`Help`** → **`Xilinx on the Web,`** then select one of the following:

- Xilinx Home Page
  The Home Page for programmable logic. From this page, you can obtain more information about Xilinx products, access online support and services, enroll in Xilinx e-Learning, buy Xilinx products online, and much more.
- LogiCORE PCI Solutions
  The Xilinx 64-bit, 66 MHz PCI-X Solution to support next generation communication systems and storage area networks.
- Support and Services
  Support and Services takes you to the Xilinx Support page at http://support.xilinx.com.
- Open a Support Case
  Allows you to open a personal WebCase with Xilinx Support.

For more information about Xilinx support available on the Web, by telephone and e-mail, see the "Technical Support" section below.

Books

Multiple printed and online books are available for use with your ISE 4.x software.

Printed Books

The following hardcopy books accompany your ISE 4.x software:

- The *ISE Installation Guide and Release Notes* describes installation procedures, key features, supported devices, and the most critical known issues.
- The *ISE Tutorial* provides an overview of the key features of the ISE 4.x software. It also contains a tutorial that demonstrates the basic design process. The tutorial includes creating HDL and schematic source files for the design, functionally simulating the design's logic, processing the design for device implementation, and using basic timing simulation to test the design in the device.

Online PDF Book Collection

The online PDF book collection is available for viewing and printing. You can access the online book collection from the Documentation CD or from the Xilinx support page on the Web at http://support.xilinx.com.

Technical Support

You can contact Xilinx for additional information and assistance in a variety of ways.

Xilinx Services and Support on the Web

Start by getting answers on the Web at http://support.xilinx.com. The support.xilinx.com Web site contains thousands of online technical solutions and product information for Xilinx software and devices. The Xilinx Answers Database is updated daily with the latest patches, problem resolutions, application notes, and data sheets.

Technical and Applications Case Support

If you cannot find your answers at support.xilinx.com, open a support case on the Web to access Xilinx Application Engineers worldwide. You can also use telephone hotlines or e-mail to open a support case.

Location	Phone	E-Mail
North America	1-800-255-7778 or 408-879-5199	Please open a case via WebCase
United Kingdom	+44-870-7350-610	eurosupport@xilinx.com
France	+33-1-34-63-01-00	eurosupport@xilinx.com
Germany	+49-89-93-08-81-30	eurosupport@xilinx.com
All other Europe locations	+44-870-7350-610	eurosupport@xilinx.com
Japan	+81-3-5321-7750	jhotline@xilinx.com
Hong Kong	(852) 2424-5200	hongkong@xilinx.com
Korea	(82) 2-761-4277	korea@xilinx.com

Note The above telephone numbers and e-mail address are subject to change. For the most current Technical Support contact information, go to http://support.xilinx.com/support/services/contact_info.htm.

When e-mailing inquiries, please provide your complete name, company name, phone number, and a complete problem description, including your design entry software and design stage. In North America, call 1-800-255-7778 if you cannot open a support case on the Web.

Training

For information on training, visit the Xilinx Support Web site at http://support.xilinx.com, or contact your local Xilinx distributor.

Get information and schedules for all courses, including recorded and live e-Learning modules. Comprehensive introductory and advanced courses cover Xilinx component and development system products.

Order Management

To order Xilinx products, visit the Xilinx Support Web site at http://support.xilinx.com, or contact your local Xilinx distributor.

Software Customer Service (Licensing)

Location	Phone	E-Mail
North America	1-800-624-4782 or (408) 879-6127	isscs@xilinx.com
Europe	Contact your local Xilinx distributor	m1license@xilinx.com
Japan	+81-3-5321-7732	cs_1@xilinx.com
All other locations	1-800-624-4782 or (408) 879-6127	cs_1@xilinx.com

Chapter 2

Project Navigator

This chapter contains the following sections:

About Project Navigator

Project Navigator is the primary user interface for ISE 4.x. You create your FPGA or CPLD design using a suite of tools accessible from Project Navigator. Each step of the design process, from design entry to downloading the design to the chip, is managed from Project Navigator as part of a project. See the "Projects" chapter of this Guide for more information.

Project Navigator Processes

Project Navigator integrates the following processes, which are described in more detail in the "Design Flow" chapter of this Guide:

- "Design Entry"
- "Constraint Entry"
- "Synthesis"
- "Simulation"
- "Implementation"
- "Device Programming"

Project Navigator Tools

Project Navigator supports the following tools:

Tool	Function	Reference
HDL Editor	create HDL files	"HDL" chapter
StateCAD and StateBench	state machine creation	"State Diagrams" chapter
Engineering Capture System (ECS)	schematic and symbol creation and editing	"Schematic Sources" chapter
LogiBLOX	design entry	"LogiBLOX" chapter
CORE Generator	design entry	"CORE Generator" chapter
Xilinx Constraints Editor	constraint entry	*Xilinx Constraints Guide*
XST (Xilinx Synthesis Technology) from Xilinx FPGA Express from Synopsys LeonardoSpectrum from Exemplar Synplify, and Synplify Pro from Synplicity	synthesis	*Xilinx Synthesis and Simulation Guide*
HDL Bencher	automated testbench or test fixture creation	*Xilinx Synthesis and Simulation Guide*
ModelSim XE / PE / SE from Model Technology Inc.	simulation	*Xilinx Synthesis and Simulation Guide*
FPGA Editor Timing Analyzer Floorplanner ChipViewer XPower	implementation	"Implementation" chapter "FPGA Implementation" chapter "CPLD Implementation" chapter
PROM File Formatter iMPACT	programming	"Device Programming" chapter

Note See the "Books" section of the "Introduction" chapter of this Guide for information on accessing the *Xilinx Constraints Guide* and the *Xilinx Synthesis and Simulation Guide*.

Starting Project Navigator

To start Project Navigator on personal computers:

- Double-click the Project Navigator icon on your desktop.

  OR

- Click **`Start`** → **`Programs`** → **`Xilinx ISE Series 4.x`** → **`Project Navigator`**.

Note Your startup menu path is created during installation, and may differ from the path shown above.

To start Project Navigator on workstations:

1. Go to the UNIX command prompt.
2. Type ise.

The first time you open Project Navigator all of its windows are empty. After that, Project Navigator opens with the last project you worked on if you selected **`Edit`** → **`Preferences`** → **`General`** → **`Window Settings`** → **`Always Open Last Project`**.

To begin using Project Navigator, you first create a project, then create or add source files for the project. To create a project, click **`File`** → **`New Project`**. See the "Projects" chapter for the steps to create a new project and to add sources.

Project Navigator Interface

The Project Navigator interface contains the following:

- "Project Navigator Main Window"
- "Sources In Project Window"
- "Processes for Current Source Window"
- "Project Workspace"
- "HDL Editor Workspace"
- "Transcript Window"

Project Navigator Main Window

The following figure shows the windows, toolbars, workspaces, and other objects in the Project Navigator main window.

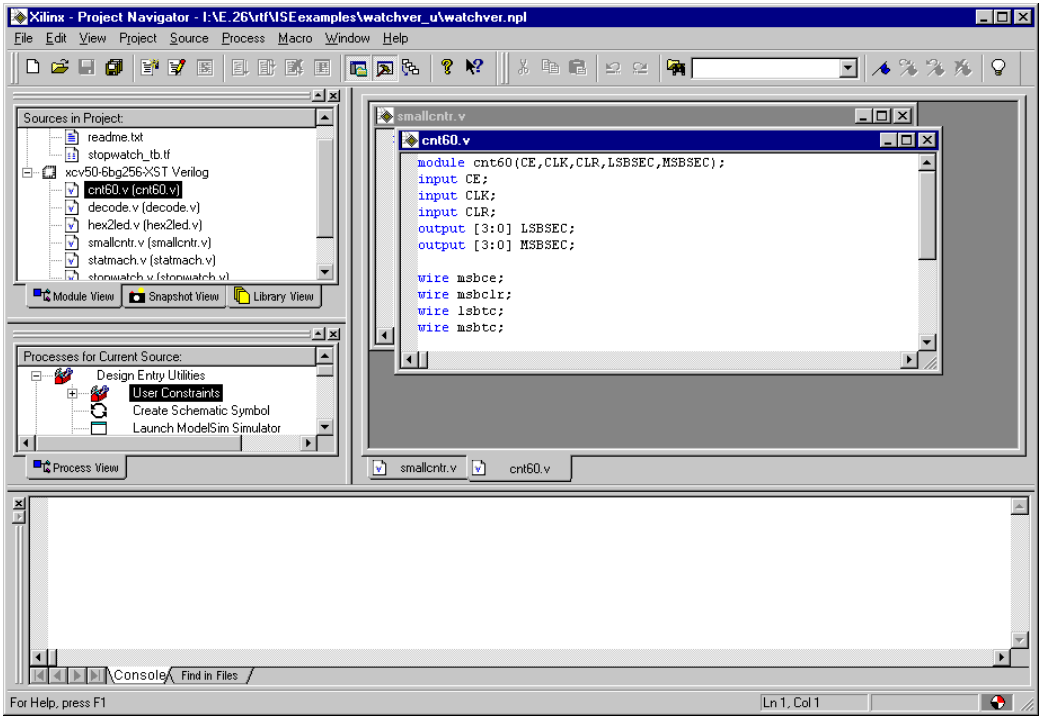

Figure 2-1 Project Navigator Main Window

The Project Navigator main window contains the following:

- Title Bar

  The Title bar displays the name of the application and the path of the current project.

- Menu Bar

  The Menu bar allows you to access the Project Navigator commands. See the Project Navigator online help for a description of all commands.

- Toolbars

  The toolbars provide convenient access to frequently used commands. There are two toolbars in the Project Navigator, a Standard toolbar and an Editor toolbar containing commands performed in the HDL Editor. See the Project Navigator online help for a detailed description of the toolbars.

- "Sources In Project Window"

  The Sources in Project window shows all the design files associated with a project. It includes tabs to view the project in Module View, Snapshot View, and Library View.

- "Processes for Current Source Window"

  The Processes for Current Source window shows the available processes for the selected source. The processes available for other sources depend upon the device and design flow you selected for the project, as well as the source type.

  Note No processes are available for user documents.

- "Project Workspace"

  The Project Workspace consists of the Sources in Project window and the Processes for Current Source window.

- "HDL Editor Workspace"

  HDL files and text files are created and edited in the HDL Editor workspace. To expand the screen area available for the HDL Editor Workspace, click **`View`** → **`Project Workspace`** to toggle the Project Workspace off.

- HDL Editor Window

  Each HDL file is opened in its own window within the HDL Editor Workspace.

- "Transcript Window"

  The Transcript window:

  - Displays informational, warning, and error messages.
  - Includes an error and warning navigation feature to help you debug your design.

- Status Bar

  The status bar displays command and processing information. Place the mouse pointer over a menu item or toolbar button to see a description of the object in the status bar.

Some tools, such as the Engineering Capture System (ECS), open in their own windows separate from Project Navigator.

Sources In Project Window

The Sources in Project window lists all the design files associated with a project. A source is any element that contains information about a design definition. Sources include:

- Files necessary to describe the behavior of your design, such as EDIF, schematics, or HDL source files
- Files needed to test your design, such as testbenches, test fixtures, and waveforms for simulation
- General project design documentation

Icons to the left of the source file names identify the type of design source.

There are two lines in the Sources in Project window when it is open in Module View:

- User Document line
- Device and Design Flow line

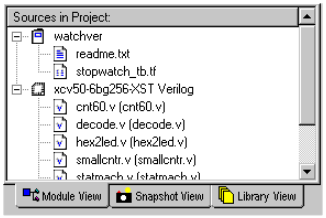

Figure 2-2 User Document Line and Device and Design Flow Line

Note Click the minus (-) icon to collapse the listings. Click the plus (+) icon to expand the listings.

User Document Line

The User Document line is the first top-level line in the module hierarchy and is identified by a document icon. It shows the project name and lists all user documents in the project. In the example above, it reads “watchver.”

User document is a source type used for any file that you want to associate with a design project, but that should not or cannot be processed by ISE 4.x. The most common use of this source type is for text documents (such as TXT, WRI, DOC) that describe the project.

Device and Design Flow line

The Device and Design Flow line is the second top-level line in the module hierarchy and is identified by a chip icon. It shows the name of a device and the design flow associated with the device. In the example above, it reads “xcv50-6bg256-XST Verilog.”

To change the Device Family, Device, and Design Flow:

1. Right-click the Device and Design Flow line.
2. Click Properties from the pull-down menu.
3. Make your changes in the Project Properties dialog box.

Menus

Use the **Project** menu to create, add, and copy project sources. The sources are created or added as described in the “Creating a Project” section of the “Projects” chapter. A source does not appear in the Sources in Project window until it is added to the project.

Use the **Source** menu to manipulate the sources listed in the Sources in Project window. Click a source to select it before accessing the Source menu. The Source menu includes the following selections:

- Open
- Close
- Rename (snapshots and VHDL libraries only)
- Remove
- Move to Library (HDL design files only)
- Properties

All source editors are linked with the Project Navigator. If a source is modified and the modification changes the hierarchy of the design, the Sources in Project window automatically updates to reflect the change.

Sources in Project Window Tabs

Tabs at the bottom of the Sources in Project window allow you to select three different views of the source data:

- "Module View"
- "Snapshot View"
- "Library View"

Module View

Click **`Module View`** to see a hierarchical representation of the design files associated with a project. These are divided into two groups in the window: user documents and project sources.

- User documents

  User documents are listed above the Device and Design Flow line. User documents do not have any processes associated with them. The Processes for Current Source window is blank when a user document is selected in the Sources in Project window.

- Project sources

  Project sources are listed below the Device and Design Flow line in a manner that depicts the relationship of these sources to each other. If you click the Device and Design Flow line or any of the sources below it, the processes that Project Navigator associates with the selected source type are listed in the Processes for Current Source window.

Both groups (project sources and user documents) behave consistently within the Sources in Project window. To view or edit a project source or user document, double-click it.

- If the source is a schematic, the associated editor is the Engineering Capture System (ECS).
- If the source is a VHDL source file, the associated editor is the HDL Editor, the Project Navigator's language sensitive text editor.

- If the user document is a text file, the associated text editor is opened.

Snapshot View

Click the **Snapshot View** tab to see a list of the snapshots you took to preserve versions of your design. Click the Source menu to open, rename, or remove a snapshot. See the "Snapshots and Archives" section below for complete information on taking and using snapshots.

Library View

Click Library View to see a list of the Libraries associated with the project and the sources included in each one. VHDL projects can include multiple libraries. See the "HDL" chapter later in this Guide for information on creating named VHDL libraries and moving modules to a library.

Source Properties

In the Sources in Project window, only the project title and the Device and Design Flow sources have properties associated with them.

To change the project title or the Device and Design Flow:

1. Highlight the project title or the Device and Design Flow line in the Sources in Project window.
2. Click **Source** → **Properties**.
3. Change the project title or Device and Design Flow in the Project Properties dialog box.

Processes for Current Source Window

The Processes for Current Source window shows all the processing tasks that apply to the module or file selected in the Sources in Project window. Project Navigator's context sensitive capability automatically determines the design flow options for a source based on the targeted device and synthesis tool. Project Navigator displays only those processes that can be performed on a specific source.

Auto-Make

Project Navigator includes an *auto-make* feature that provides results-oriented processing. You determine the end result of the processing by selecting a process in the Processes for Current Source window. When you click a process, the auto-make feature checks for dependencies between the process you selected and any predecessor processes that may be out of date or that may not have been run.

Auto-make automatically runs the necessary processes to bring your design up-to-date so that it can complete the process you requested. Auto-make reduces design errors by ensuring that each process step operates on the most current process results and design data.

Setting Properties for Processes

Design processes execute other programs or sets of programs. Very often you can set parameters to be passed into these programs. For example, you can set a parameter to insert I/O buffers during synthesis, or one to use zero or maximum delays for a simulation. The most common properties for a given process are selected as the defaults. You do not need to set the process properties to run a process.

To specify properties for a process:

1. Highlight the process.
2. Click **`Process`** → **`Properties`**.

   The Process Properties dialog box opens containing parameters appropriate for the process.

If no properties can be set for that process, the Properties selection is grayed out. If the Property Name in the dialog box cannot be fully viewed or appears truncated, use the grabber bar to expand the Property Name field. Click OK to preserve the new field length settings.

Viewing Reports

In many cases, a report is the output of a process. To view a report, double-click its name in the Processes for Current Source window. If the report does not currently exist, it is generated. A green check mark by the report name indicates that the report is up-to-date, and no processing is performed.

If the report is not up-to-date, to update the report before you view it:

1. Click the report name.
2. Click **Process** → **Run**.

   The auto-make process automatically runs only the necessary processes to update the report before displaying it.

Click **Process** → **Rerun All** to re-run all processes—even those processes that are currently up-to-date—from the top of the design to the stage where the report would be generated before displaying the report.

Reports on the synthesis process vary depending on the synthesis tool you are using. Reports for the Implement Design process vary depending on whether the target device is an FPGA or a CPLD. See the "Implementation" chapter for descriptions of the implementation reports.

Project Workspace

The Project Workspace consists of the Sources in Project window and the Processes for Current Source window. These windows are grouped together for viewing purposes. Click **View** → **Project Workspace** to toggle the display of these two windows as one item.

HDL Editor Workspace

The HDL Editor workspace is the main text editing area for HDL code. It is the only Project Navigator window that cannot be hidden or undocked (see the"Docking and Undocking Windows and Toolbars" section below).

Tabs at the bottom of the HDL Editor workspace allow easy access to the file you want to view or edit. The Window menu provides standard window functions, such as cascading and tiling, for managing open windows.

Two toolbars are available in Project Navigator:

- The Standard toolbar (left side of the toolbar area)
- The Editor toolbar (right side of the toolbar area). The Editor toolbar is used exclusively with the files in the HDL Editor workspace.

See the online help for information on using the toolbars. Use the View menu to toggle the toolbars on and off.

Transcript Window

The Transcript window contains a project log. The output of all processes is captured here, including error and warning messages from the synthesis and implementation tools.

If the error or warning was generated by the synthesis tool (XST or FPGA Express), you can go to the line in the VHDL or Verilog source file containing the error.

For all errors and warnings, you can use the search engine on the Xilinx Web site to find a Solution Record pertaining to your problem. For more information on navigating to a source file and navigating to a solution record, see the Transcript window online help.

Customizing Project Navigator

You can customize Project Navigator by:

- "Setting Display Preferences"
- "Displaying and Hiding Windows and Toolbars"
- "Docking and Undocking Windows and Toolbars"

Setting Display Preferences

Click **`Edit`** → **`Preferences`** to open the Preferences dialog box. The Preferences dialog box contains four tabs:

- "General"
- "Editor"
- "Processes"
- "Partner Tools and Web Browser"

General

From the General tab you can set:

- Window settings, including Always Open Last Project and Use File Associations on User Documents (personal computers only)
- The default path for new sources, including Relative Paths and Absolute Paths
- The project font (the font used in the Sources in Project and Processes for Current Source windows)

Editor

From the Editor tab, you can select:

- Attributes for tabs
- The font used in HDL Editor windows

Processes

From the Processes tab, you can specify:

- "Property Display Level"
- "Process Tree Default"

Property Display Level

You can set process properties for processes such as simulation, synthesis, or implementation. The available properties are displayed in a Process Properties dialog box. You control whether to include additional advanced properties in the Process Properties list. By default, only the standard properties are listed.

Standard displays all of the Standard properties and their default values in the Process Properties dialog boxes. Advanced displays all of the Standard properties as well as any Advanced properties and their default values in the Process Properties dialog boxes.

Process Tree Default

Determines the default hierarchical display of processes in the Processes for Current Source window. You can choose to have all processes expanded or collapsed.

Partner Tools and Web Browser

The display of the Partner Tools and Web Browser tab depends on whether you are running ISE on a personal computer or UNIX.

Partner Tools (Personal Computers Only)

The Partner Tools tab allows you to specify the location of third party executables.

Web Browser (UNIX Only)

The Web Browser tab allows you to specify the location of the browser used to view Web pages. By default, it is set to Netscape.

Displaying and Hiding Windows and Toolbars

Click **View** to control the windows and toolbars you want to display:

Menu Item	Function
Standard Toolbar	Displays or hides the Standard toolbar.
Editor Toolbar	Displays or hides the Editor toolbar.
Project Workspace	Displays or hides the Project Workspace (the Sources in Project window and the Processes for Current Source window)
Sources	Displays or hides the Sources in Project window
Processes	Displays or hides the Processes for Current Sources Window
Transcript	Displays or hides the Transcript window
File Names	Displays or hides the display of file names in the Module View of the Sources in Project window

Menu Item	Function
Refresh F5	Rebuilds the project hierarchy and updates the Sources in Project window
Minimize All Windows	Minimizes all ISE Windows

To hide the window or toolbar, right-click to display the popup-menu, then click **Hide** (just below **Allow Docking**). To redisplay it, click that window or toolbar from the **View** menu. The **Hide** selection is on the source or process window context menu.

Docking and Undocking Windows and Toolbars

Project Navigator allows you to dock and undock most windows and toolbars. You can remove windows and toolbars from the Project Manager interface and place and size them separately. See the Project Navigator online help for detailed information on docking and undocking windows and toolbars.

Snapshots and Archives

To save a specific revision of your project, you can archive it or take a snapshot of it.

When you archive a project, you create a ZIP file for that version and place it in a specified directory. To archive a project, click **Project** → **Archive** in Project Navigator. To open an archive or see the files in it, unzip it.

When you take a snapshot of a project, the snapshot becomes part of the project. It is accessible from the Snapshot View of the Sources in Project window. You can open it at any time to view its contents. If you replace the current project with a snapshot, you can make changes and reprocess the snapshot version of the project as desired. You can also replace the current version of a project with one captured previously as a snapshot.

Taking a Snapshot

To take a snapshot of the current version of your project:

1. Click **Project** → **Take Snapshot**.
2. Enter a name and comments in the Take a Snapshot of the Project dialog box.

   Note The default name is *snap1*. Snapshot names cannot contain spaces. The name appears in the Snapshot View of the Sources in Project window. Comments appear after the name in the Sources in Project window if the Files Names command in the View menu is enabled.

The snapshot is saved in a separate directory (specified by the Snapshot Name) under the project's snapshot directory to isolate it from the present working project files. All the project's snapshots are listed on the Snapshot View tab in the Sources in Project window.

All source files that have been added to the project (including user documents) are copied for the snapshot. The process files necessary to recreate the state of the project when you made the snapshot are also copied.

Renaming a Snapshot

To rename a snapshot:

1. Go to the Sources in Project window.
2. Click the Snapshot View tab.
3. Click a snapshot name.
4. Click **Source** → **Rename** from the Project Navigator.
5. Modify the name. The selected snapshot name will change so you can edit it.

Editing a Snapshot Comment

To edit a snapshot comment:

1. Go to the Sources in Project window.
2. Click the Snapshot View tab.
3. Click a snapshot name.

4. Click **Source** → **Rename** from the Project Navigator.
5. Edit the comment.

   Note You cannot add a comment after creating a snapshot. You can only edit one that was entered when the snapshot was created.

Deleting a Snapshot

To delete a snapshot:

1. Go to the Sources in Project window.
2. Click the Snapshot View tab.
3. Click a snapshot name.
4. Click **Source** → **Remove** from the Project Navigator.
5. Confirm the deletion of all the files.

The snapshot and all its associated files are deleted.

Viewing Snapshot Contents

To open and view the files contained in a snapshot:

1. Go to the Sources in Project window.
2. Click the Snapshot View tab.
3. Click a snapshot name.
4. Click **Source** → **Open** from the Project Navigator.
5. The sources for the selected snapshot appear in a hierarchical display under the snapshot name in the Sources in Project window. Highlight a source to display the processes for that source (with check marks status) in the Processes for Current Source window.

Note The snapshot is read-only. You can view the files and process status but you cannot change it unless you make it the current project as described in the "Snapshots and Archives" section.

Viewing Snapshot Source Files and Reports

When the hierarchical contents of a snapshot are displayed in the Sources in Project window, double-click a source to open the file in the HDL Editor workspace or its associated text editor. Snapshot source files in the HDL Editor workspace are read-only files.

You can also double-click a report to open it as if it were a source file. Because you can have multiple reports open at once, you can open reports from multiple snapshots for comparison.

Replacing the Current Project with a Snapshot

To make changes to the project version represented in a specific snapshot, make that snapshot version the current version of the project.

> **Caution** This procedure replaces the current version of the project with the version saved in a snapshot. Your current project will no longer exist unless you take a snapshot of it *before* you replace it with a previous snapshot.

To replace the current version of a project with a previous snapshot:

1. Go to the Sources in Project window.
2. Click the Snapshot View tab.
3. Click a snapshot name.
4. Click **`Project`** → **`Replace with Snapshot`** from the Project Navigator.
5. Click **`Yes`** to save your current project as a snapshot. Click **`No`** to replace your current project without saving it as a snapshot.

   If you click **`Yes`**, the Taking a Snapshot of the Project dialog box opens to allow you to enter a Snapshot Name and optional comments for the snapshot of the current project.

The current project is immediately replaced in the Module View with the version represented in the selected snapshot. When a snapshot becomes the current project, the snapshot version remains in the snapshot directory and is not overridden by changes made to its current project version.

Note To cancel this process:

1. Click Yes from the first dialog box.
2. Click Cancel on the Taking a Snapshot of the Project dialog box.
3. Click No on the next dialog box.

Restoring a Snapshot or Archive with Remote Sources

A remote source file is a source file which resides outside the main project directory. When a snapshot or archive is created for a project which includes remote sources, the remote source files are copied into the snapshot or the archive the same way a local source file would be. However, when you make a snapshot current, or restore an archived project, these remote source files are NOT copied back to the remote location. Rather, they are copied into a subdirectory called *remote_sources* in the main project directory. The project still maintains the links to the remote location of the source files.

To continue working with the snapshot or archived versions of the source files, manually copy them from the *remote_sources* subdirectory back to the original remote location. The original locations of the remote source files are still represented in the Source Hierarchy displayed in the Source View window in the Project Navigator (remote sources are always displayed with their full pathnames).

For example, you have a project in C:\MyProject which contains a remote source file named foo.vhd located at C:\MySourceFiles\foo.vhd. When you create a snapshot for this project, the file C:\MySourceFiles\foo.vhd is copied into the snapshot along with the rest of the design files. When you make the snapshot current by selecting **`Make Snapshot Current`**, the foo.vhd file is copied to C:\MyProject\remote_sources\foo.vhd. To continue working with this file, copy it from C:\MyProject\remote_sources\ to C:\MySourceFiles\.

Using the same example, if you archive C:\MyProject, the remote sources are copied to C:\MyProject\remote_sources, and the archive is then performed. When you restore the archive from MyProject.zip, the remote sources are restored to the C:\MyProject\remote_sources directory. Copy them to the remote locations if desired.

Chapter 3

Projects

This chapter contains the following sections:

About Projects

ISE 4.x organizes and tracks your FPGA or CPLD design as a *project*. This section discusses some of the key concepts and features of projects.

- Project

  A project is a collection of all files necessary to create and download your design to the selected device.

- Project Properties

  Each project has a directory, device family, device, and design flow associated with it as *project properties*. They enable Project Navigator to display and run only those processes appropriate for the targeted device and design flow. See the "Setting Project Properties" section below.

- Sources

  All projects have *sources*. A *source* is any element that contains information about a design, such as HDL files, state diagrams, schematics, documentation files, simulation models, and test files. You create and add the sources to your project. See the "Sources" and "Source Types" sections in this chapter.

- Process

  A *process* is an action you can perform on a source to test and implement your design.

- Project Management

  Project Navigator manages your project based on the target device and design flow you selected when you created the project. It organizes all the parts of your design, and keeps track of the processes necessary to move the design from the conceptual stage through to implementation in the targeted Xilinx device.

- Auto-Make

  Project Navigator's auto-make feature monitors dependencies between process steps, and automatically runs and updates the intermediate processes when necessary. See the "Auto-Make" section of the "Project Navigator" chapter.

- Context Sensitivity

  Project Navigator is context sensitive with respect to source file types, device types, and design flow selection. The steps listed in the Processes for Current Source window reflect this context sensitivity. For example, if you highlight a text file describing the project, there is no processing to perform, and the Processes for Current Source window is blank.

  In contrast, if you select a top level module, all the processes that can be performed from that point in the design hierarchy are listed in the Processes for Current Source window.

Creating a Project

Creating a project consists of the following:

- "Setting Project Properties"
- "Specifying a Project Name and Directory"
- "Selecting a Device and Design Flow"
- "Supported Devices and Design Flows"
- "Project Flow Characteristics"

Setting Project Properties

Specify the following project properties when you create a new project:

- Project name

  Give your project a unique name.

- Project directory

  Specify a unique directory to store the project's source files, intermediate data files, and resulting files.

- Device family

  Target your design for a specific Xilinx device family (architecture). Virtex, Spartan, XC9500 are examples of device families.

- Device

  Target your design for a specific device within the selected device family. An example of a device is *S05 PC844-4*, where *S05* designates a specific Spartan device, *PC844* specifies the package, and *-4* indicates the speed grade.

- Design flow

  Specify a design flow, either EDIF or a synthesis tool. FPGA Express and XST are the synthesis tools. VHDL, Verilog, and ABEL versions are available for these tools.

Specifying a Project Name and Directory

Specify a project name and directory when you create a new project. For each project, designate a separate, unique directory containing only one project file (*project_name*.npl).

To create a new project:

1. Click **`File`** → **`New Project`**.
2. In the New Project dialog box, enter a name for the project in the Project Name field.

   The Project Navigator uses the name entered here to create the project file (*project_name*.npl). The name you type in the Project

Name field appears as a subdirectory in the Project Location field.

The project targets the default device and uses the design flow identified in the Project Device and Design Flow field of the New Project dialog box. See the "Selecting a Device and Design Flow" section below.

Selecting a Device and Design Flow

The first time you create a new project, a default device and design flow appear in the New Project dialog box. For subsequent projects, the device and design flow you used for your last project are used as the default for the new project. Choose the device and design flow in the New Project dialog box when you create the project. You can change them later.

Selecting a Device

To select or change a device:

1. Open the project.
2. Click the Device and Design Flow line in the Sources in Project window.
3. Click **`Source`** → **`Properties`**.

   The Project Properties dialog box opens with the current values identified in the Value fields.
4. Click the scroll button on the right side of the Device Family Value field to display the device family list.
5. Scroll through the device family list and select the device by clicking its name.
6. When you select a device family, the Project Properties dialog box automatically updates with a default device and appropriate design flow for that device family.

   For example, if you select Spartan for the Device Family, the default Spartan device (S05 PC84-4) appears in the Device Value field. The device information includes the device name (S05), device package (PC84), and speed (-4).) FPGA Express VHDL appears as the default Synthesis Tool for Spartan devices.

7. Modify the defaults by placing the cursor on the right side of a Value field and scrolling through the list that appears.

Selecting a Design Flow

The selection of a design flow is closely linked to the Xilinx device family and the type of design (VHDL or Verilog) you chose for your project. It also depends on the synthesis tools that are installed. You can change the design flow at any time.

> **Caution** Changing the design flow after initial selection may require changing the targeted device.

To select or change a design flow:

1. Open the project.
2. Click the Device and Design Flow line in the Sources in Project window.
3. Click **`Source`** → **`Properties`**.

   The Project Properties dialog box opens with the current values shown in the Value fields.
4. Click the right side of the design flow value field to display a list of the design flows supported for the selected Device Family.
5. Scroll through the design flow list and select the tool by clicking its name. Only design flows that are appropriate for the selected device family and that are installed are listed.

Supported Devices and Design Flows

For a list of the most current supported device families and device flows, see the Device Support table included with the ISE online help (**`Help`** → **`ISE Help Contents`** → **`Reference`** → **`Device Support).`**

Note Xilinx supports LeonardoSpectrum and Synplify, but does not provide these synthesis tools with the ISE 4.x software. These are third party tools that you must purchase and install separately from ISE.

Project Flow Characteristics

The following sections briefly describe the project flow characteristics for each design flow. Not all devices support all project flows. For a list of the devices supported by each project flow, click **Help** → **ISE Help Contents** → **Device Support.**

XST VHDL

- Uses Xilinx Synthesis Technology (XST) for synthesis
- Can contain VHDL code only. No ABEL-HDL is allowed
- Creates a functional VHDL model for schematics prior to synthesis

XST Verilog

- Uses Xilinx Synthesis Technology (XST) for synthesis
- Can contain Verilog code only. No ABEL-HDL is allowed
- Creates a functional Verilog model for schematics prior to synthesis

FPGA Express VHDL

- Uses FPGA Express from Synopsys for synthesis
- Can contain mixed VHDL and Verilog code. No ABEL-HDL is allowed.
- Creates a functional VHDL model for schematics prior to synthesis
- Can include the Express Constraints Editor, Express Time Tracker, and Schematic Viewer tools from Synopsys in certain configurations

FPGA Express Verilog

- Uses FPGA Express from Synopsys for synthesis
- Can contain mixed VHDL and Verilog code. No ABEL-HDL is allowed.
- Creates a functional Verilog model for schematics prior to synthesis

- Can include the Express Constraints Editor, Express Time Tracker, and Schematic Viewer tools from Synopsys in certain configurations

ABEL XST VHDL

- Uses Xilinx Synthesis Technology (XST) for synthesis
- Can contain only ABEL-HDL code and VHDL testbenches. No Verilog is allowed.
- Creates a functional VHDL model for ABEL-HDL code prior to synthesis
- Creates a functional VHDL model for schematics prior to synthesis

ABEL XST Verilog

- Uses Xilinx Synthesis Technology (XST) for synthesis
- Can contain only ABEL-HDL code and Verilog test fixtures. No VHDL is allowed.
- Creates a functional Verilog model for ABEL-HDL code prior to synthesis
- Creates a functional Verilog model for schematics prior to synthesis

LeonardoSpectrum VHDL

- Can contain VHDL code only. No ABEL VHDL allowed
- Creates an EDIF netlist

LeonardoSpectrum Verilog

- Can contain Verilog only. No ABEL-HDL allowed.
- Creates an EDIT netlist

Synplify VHDL

- Can contain mixed VHDL and Verilog code. No ABEL-HDL allowed.
- Creates an EDIF netlist

Synplify Pro VHDL

- Can contain mixed VHDL and Verilog code. No ABEL-HDL allowed.
- Creates an EDIF netlist

Synplify Verilog

- Can contain mixed VHDL and Verilog code. No ABEL-HDL allowed.
- Creates an EDIF netlist

Synplify Pro Verilog

- Can contain mixed VHDL and Verilog code. No ABEL-HDL allowed.
- Creates an EDIF netlist

Sources

A *source* is any element that contains information about a design. After you create your project and select a device and design flow, you can begin creating and adding source files. A project may contain the following sources:

- The logical descriptions for programming the selected Xilinx device, including:
  - HDL files
  - State diagrams
  - Schematics
- Documentation files
- Simulation models
- Test files (testbenches or test fixtures)

The types of sources (schematic, VHDL, Verilog, ABEL) available in a project vary depending on the selected device and design flow. See the "Source Types" section below for information on the types of sources supported for ISE 4.x projects.

Creating a New Source

To create a new source file:

1. Open the project.
2. Click **`Project`** → **`New Source`**.

   The New dialog box opens with a list of appropriate sources for the project's target device and design flow.
3. Select the source type you want to create from the list.
4. Enter a name for the new source file in the File Name field.
   - Do not add an extension to the file name. ISE 4.x adds the appropriate extension for the selected source type.
   - Sources cannot have spaces or periods in their names.
   - State diagram names are limited to eight characters and must start with an alphabetic character.
5. Check the Add to Project box to add this source automatically after it is created.

   Note The Add to Project box does not apply to all source types. You must manually add certain sources, such as state diagrams, as described below in the "Adding an Existing Source" section. Testbenches and test fixtures are added automatically.
6. Click **`Next`**.

   The New Source Information window opens to summarize the requested new source.

   Note If you are adding a VHDL Module or Verilog Module, a source wizard opens before the New Source Information window. See the "HDL Sources" section of the "HDL" chapter for information on using the HDL source wizard.
7. For the following source types, the Select window will open. You are asked to select a source from a list of sources in the project to associate the new source with.
   - test fixture
   - testbench

- BMM file
- testbench waveform

8. Click **Next** in the Define Sources in Project window.
9. Click **Finish** in the New Source Information window.

The source type you selected determines the next step. For example, if you are creating a schematic, the Engineering Capture System (ECS) opens. If you are creating a user document, a text editor opens.

A source file with the name specified in the File Name field is loaded in the selected source tool ready for you to create the source. If you checked Add to Project, the source is automatically added and is listed in the Sources in Project window.

Adding an Existing Source

You can add an existing source to a project. The source file can reside in the project directory or in a remote directory.

To add an existing source file:

1. Open a project.
2. Click a source in the Sources in Project window.
3. Click **Project** → **Add Source**.
4. Use the Add Existing Sources in Project dialog box to browse through your directories and select the source you want to add. The source you select remains in its current directory. It is not moved or copied to the project's directory.
5. Click open.
6. After selecting the file, the Choose Source Type dialog box may open. If so, identify the file type you are adding. For example, a file with the .vhd extension can be a VHDL Module, testbench, or package.

When you click **Open** to select a file in the Add Existing Sources in Project dialog box or **OK** in the Choose Source type dialog box, the file appears in the Sources in Project window for the current project.

The directory path appears along with the file name for all remote sources (sources not in the current project directory). To truncate or

expand the path information, click **Project** → **Toggle Paths** from the Project Navigator menu.

If the added source references other sources that are not currently in the project, a red question mark appears beside the undefined source.

Adding a Copy of an Existing Source

You can add a *copy* of an existing source to a project. The source can be in the project directory, or in a remote directory.

- If the source is in the project directory, a second copy is not made. The original is added to the project.
- If the source is in a remote directory, a copy of the source is created and placed in the project directory. The copy in the project directory is added to the project.

To add a copy of an existing source to a project:

1. Click **Project** → **Add Copy of Source.**
2. Follow the procedures shown above in "Adding an Existing Source".

Source Types

Projects can include the source types listed in the following table.

Table 3-1 Permissible Source Types

Source Type	File Extension
Project file	.npl
User document (such as a specification)	.txt, .wri, .doc, .xls, .hlp (or any other extension not recognized by Project Navigator)
Schematic	.sch
State diagram	.dia
VHDL module	.vhd
VHDL package	.vhd
VHDL testbench	.vhd
Waveform stimulus	.wdl

Table 3-1 Permissible Source Types

Source Type	File Extension
Verilog module	.v
Verilog test fixture	.tf
ABEL-HDL logic description	.abl
ABEL-HDL test vectors	.abv (or .abl)
COREgen IP	.xco
LogiBLOX module	.mod
Testbench Waveform	.tbw
Block Memory Map file	.bmm
Executable CPU code image file	.elf
EDIF Source file	.edn, .edf, .edif, .sedif

Synthesis Tool Support

Table 3-2 Synthesis Tool Support

Source Type	Synthesis Tool					
	XST VHDL	XST Verilog	FPGA Express	ABEL	Leonardo Spectrum	Synplify
User document	X	X	X	X	X	X
Schematic	X	X	X	X	X	X
VHDL Module	X		X		X	X
VHDL Testbench	X		X		X	X
VHDL Package	X		X		X	X
VHDL Library	X		X	X	X	X
Verilog Module		X	X		X	X
Verilog Test Fixture		X	X		X	X
ABEL HDL Module (CPLD only)				X		
ABEL Test Vector (CPLD only)				X		
State Diagram	X	X	X	X	X	X
COREgen IP (FPGAs only)	X	X	X		X	X
LogiBLOX Module	Spartan, SpartanXL, CPLD	Spartan, SpartanXL, CPLD	XC 4000, Spartan, SpartanXL, CPLD	CPLD	XC 4000, Spartan, SpartanXL, CPLD	XC4000

Source Type Descriptions

ISE 4.x supports the following source types:

- "State Diagram"
- "Schematic"
- "VHDL Module"
- "VHDL Testbench"

- "VHDL Package"
- "VHDL Library"
- "Verilog Module"
- "Verilog Test Fixture"
- "Testbench Waveform"
- "ABEL-HDL Module (CPLDs Only)"
- "ABEL Test Vector (CPLDs Only)"
- "CORE Generator Module"
- "LogiBLOX Module"

The following sections describe the source types you can add to your project. All sources appear in the Project Navigator Sources in Project window. See the "Sources In Project Window" section of the "Project Navigator" chapter for detailed information on how to access and display source data.

State Diagram

Drawing a state diagram is one method you can use to define your design. A state diagram is a graphical representation of a finite state machine. The state diagram source file (.dia) can be added as a user document. You can add the optimized HDL module (.vhd or .v) translated from the state machine as a VHDL or Verilog source file. After the state diagram file and its corresponding HDL file are added, they are updated whenever modifications are made to either of them using the state diagram tool.

ISE 4.x includes support for StateCAD for the creation and development of state machines and their translation to HDL code.

Schematic

Schematics (.sch) are another form of design entry. Schematic diagrams are created in the Engineering Capture System (ECS) and automatically added to your project. Schematic sources are automatically translated into VHDL or Verilog modules for simulation and synthesis. The VHDL or Verilog functional modules are not listed in the Sources in Project window.

To view the functional module:

1. Click a schematic source.
2. Click View VHDL (or Verilog) Functional Model in the Processes for Current Source window.

VHDL Module

A VHDL module (.vhd) is a source file that contains a single VHDL entity and architecture pair. The architecture should be synthesizable VHDL. You can create a VHDL module using the HDL Editor. See the "HDL" chapter for information on creating VHDL source files.

VHDL Testbench

A VHDL testbench (<*vhdl_modulename*>_tb.vhd) is a source file containing a single entity and architecture pair that provides the stimulus for another VHDL design unit during simulation. In ISE 4.x, VHDL Testbench sources are associated with the source file that they instantiate. To enable ISE 4.x to automatically launch a simulation using the installed simulator, the entity name of any testbench source must be "testbench."

A VHDL testbench is easy to recognize because the entity declaration has no ports. It is the entire universe to the unit under test. Nothing may enter or leave it.

Testbench Waveform

Use the HDL Bencher to edit this file.

VHDL Package

VHDL models may be defined using packages. Packages contain type and subtype declarations, constant definitions, function and procedure definitions, and component declarations.

XST also supports predefined packages; these packages are pre-compiled and can be included in VHDL designs. These packages are intended for use during synthesis, but may also be used for simulation. See the *Xilinx Synthesis Technology (XST) User Guide* for a list and description of supported predefined packages.

VHDL Library

VHDL requires all design sources to be in a library. See the "HDL" chapter for information on creating and naming VHDL libraries.

Verilog Module

A Verilog module (.v) is a file that contains code for a single Verilog module. See the "HDL" chapter for information on creating Verilog source files.

Verilog Test Fixture

A Verilog test fixture (.tf) is a file containing a single module that provides the stimulus for another Verilog design unit during simulation. In ISE 4.x, Verilog test fixture sources are associated with the source file that they instantiate.

ABEL-HDL Module (CPLDs Only)

An ABEL-HDL module (.abl) is a file containing ABEL code. See the "HDL" chapter for information on creating HDL source files.

ABEL Test Vector (CPLDs Only)

An ABEL test vector (.abv or .abl) is a file containing a single module that provides the test vectors necessary to simulate your design.

CORE Generator Module

A CORE Generator module is a module from the CORE Generator library or one customized with the CORE Generator tool. The CORE Generator delivers parameterizeable cores optimized for Xilinx FPGAs. It provides a catalog or ready-made functions ranging in complexity from simple arithmetic operators such as adders to system-level building blocks that include filters and memories.

LogiBLOX Module

A LogiBLOX module is a module from the LogiBLOX library of generic modules or one customized with the LogiBLOX tool. LogiBLOX modules are high-level modules such a counters, shift

registers, and multiplexers that are pre-optimized for XC4000, Spartan, SpartanXL, and CPLD devices.

See the "LogiBLOX" chapter for information on using LogiBLOX in your project.

Chapter 4

Design Flow

This chapter contains the following sections:

About Design Flow

Design flow is a multi-step, iterative process that includes:

- "Design Entry"

  Create your design using HDL code, schematics, Intellectual Property such as CORE Generator, and state diagrams.

- "Constraint Entry"

  Enter timing, placement, and other constraints in your design at various stages, using a variety of tools and methods.

- "Synthesis"

  Translate your design into gates and optimize it for the target architecture.

- "Simulation"

  Verify the operation of your design before you implement it as hardware.

- "Implementation"

  Convert the logical design file format (EDIF or NGO) created during design entry into a physical file format for a specific Xilinx architecture.

- "Device Programming"

  Create a programming file that can be downloaded to the target device.

Design Entry

You can create your design using HDL code, Intellectual Property such as CORE Generator, schematics, and state diagrams. You can create new sources or add existing sources to your project. Behavioral simulators are available to test the logic of your designs before continuing to the next stages.

ISE 4.x includes the following design entry tools, all of which are accessible from Project Navigator:

- "HDL Editor"
- "StateCAD State Machine Editor"
- "Engineering Capture System (ECS)"
- "CORE Generator"
- "LogiBLOX"

HDL Editor

HDL Editor is a language-sensitive text editor for VHDL, Verilog, and ABEL- HDL. HDL Editor is integrated into Project Navigator.

Note ABEL-HDL is supported for CPLDs only.

HDL Editor includes color coding and context sensitive help for reserved words. The Project Navigator New Source Wizard can build the initial text structure for your HDL file. Project Navigator also includes a Language Template feature with pre-built language and synthesis templates to assist with HDL entry. For more information, see the "HDL" chapter later in this Guide.

StateCAD State Machine Editor

Use StateCAD® and StateBench® to create state diagrams for state machine designs. The Project Navigator launches StateCAD for source creation or modification. You can add State diagrams and their corresponding StateCAD-generated HDL source modules to your project. Once added, they are updated automatically in Project Navigator whenever they are modified within StateCAD. Use the StateBench simulator to verify the behavior of your state diagram. See the "State Diagrams" chapter later in this Guide.

Engineering Capture System (ECS)

For schematic designs, Project Navigator launches the Engineering Capture System (ECS). ECS includes both a schematic editor and a symbol editor. The schematic editor provides a graphical entry method to capture designs. The symbol editor allows you to create and customize a variety of electrical symbol types. See the "Schematic Sources" chapter later in this Guide.

CORE Generator

CORE Generator is a design tool that delivers parameterized cores optimized for Xilinx FPGAs. It provides you with a catalog of ready-made functions ranging in complexity from simple arithmetic operators such as adders, accumulators, and multipliers, to system-level building blocks.

The CORE Generator System creates customized functional building blocks such as FIR filters, FIFOs, and multipliers, and delivers high levels of performance and area efficiency. This is accomplished by taking advantage of Xilinx's core-friendly FPGA architectures and Xilinx Smart-IP™ technology.

Xilinx Smart-IP technology provides FPGA architectural advantages such as look-up tables (LUTs), distributed and block RAM, embedded multipliers, and segmented routing. This technology also enables relative location constraints, expert logic mapping, and floorplanning to optimize performance of a given core instance in a given Xilinx FPGA architecture.

The CORE Generator System produces a Electronic Data Interchange Format (EDIF) netlist, a Verilog Output (VEO) with a Verilog (V) wrapper file or, VHDL Output (VHO) template file with a VHDL (VHD) wrapper file. The Electronic Data Netlist (EDN) file contains the information for implementing the module in a Xilinx FPGA. The .ASY, .VHX and .XSF symbol information files allow you to integrate the CORE Generator module into a schematic design for ECS, Innoveda, Mentor, or Foundation tools. Finally, the VEO and VHO template files contain code that can be used as a model for instantiating a CORE Generator module in a Verilog or VHDL design.

For detailed information, see the *CORE Generator Guide* and the "CORE Generator" chapter later in this Guide.

LogiBLOX

ISE 4.x includes LogiBLOX to aid in the creation of high-level modules, such as shift registers and counters. LogiBLOX supports XC4000, Spartan, and SpartanXL devices only. See the "LogiBLOX" chapter later in this Guide.

Constraint Entry

Constraints are instructions placed on components or nets through a user constraint file (UCF), HDL code, or schematic. Constraints can indicate such things as placement, implementation, naming, directionality, and timing. For detailed information about constraints, see the *Xilinx Constraints Guide*.

Constraint Types

The following constraints are available for use with your FPGA and CPLD designs:

- Timing Constraints
- Placement Constraints
- Grouping Constraints
- Mapping Directives
- Routing Directives
- Modular Design Constraints

- Synthesis Constraints
- Fitter Directives
- Initialization Directives
- DLL and DCM Constraints
- Logical and Physical Constraints

Caution Not all constraints are available for all devices, nor can they be entered with every tool or method. For details, see the *Xilinx Constraints Guide*, especially Appendix A which lists every constraint and specifies its type, the devices with which it is available, and the methods by which it can be entered.

Third Party Constraints

A third party constraint is a constraint from a company other than Xilinx that is supported within the Xilinx technology. Xilinx supports third party constraints from the following companies:

- Synplicity
- Synopsys
- Exemplar

For more information about third party constraints, see the *Xilinx Constraints Guide.*

Constraint Entry Tools

Constraints can be entered at various stages throughout the entire design process, using a variety of tools. Constraint entry methods and tools include the Xilinx Constraints Editor, UCF files, FPGA Express, and XST Constraint files, as well as several others. For more information about using each of these tools and methods to enter constraints, see the *Xilinx Constraints Guide*.

Synthesis

After your design has been successfully analyzed, the next step is to translate the design into gates and optimize it for the target architecture. This is the synthesis phase. Two synthesis tools are included on the ISE 4.x CD-ROM:

- "XST from Xilinx"
- "FPGA Express from Synopsis"

ISE 4.x also provides integration with the following synthesis tools:

- "Synplify from Synplicity"
- "LeonardoSpectrum from Exemplar"

You choose the synthesis tool in the Design Flow option when you create a project. For detailed information about synthesis, see the *Simulation and Synthesis Design Guide.*

XST from Xilinx

XST (Xilinx Synthesis Technology) is a Xilinx tool that synthesizes HDL designs to create an NGC file. The Project Navigator invokes XST processing when you select a source and then select a synthesis process for a project that has the XST synthesis tool associated with it. An XST flow project can contain *either* VHDL (XST VHDL) *or* Verilog (XST Verilog) modules, but not a mix of both. A functional VHDL model (XST VHDL) or Verilog model (XST Verilog) is created for schematics prior to synthesis. Process properties can be set to control XST synthesis.

FPGA Express from Synopsis

FPGA Express from Synopsys, Inc., can synthesize VHDL, Verilog, or mixed HDL designs to create EDIF netlists. The Project Navigator invokes FPGA Express processing when you select a source and then select a synthesis process for a project that has the FPGA Express synthesis tool associated with it.

Depending on the ISE 4.x configuration you purchased, the Express Constraints Editor (pre-optimization), and Time Tracker (post-optimization) GUIs may also be available to you. A functional VHDL model (FPGA Express VHDL) or Verilog module (FPGA Express

Verilog) is created for schematics prior to synthesis. Process properties can be set to control FPGA Express synthesis.

Both FPGA Express VHDL and FPGA Express Verilog support mixed HDL designs. The designation VHDL or Verilog when you select an FPGA Express synthesis tool refers to whether Verilog or VHDL functional models are created for schematics.

For general information, see the Synopsis Web site at http://www.synopsys.com/.

Synplify from Synplicity

Synplify is a third party synthesis tool from Synplicity, Inc., that can effectively synthesize VHDL, Verilog, and mixed language designs to create EDIF netlists. ISE 4.x works with Synplify and Synplify Pro 6.2 and higher. For general information, see the Synplicity Web site at http://www.synplicity.com/. For detailed information, see the Synplify documentation, or the online help provided with Synplify.

LeonardoSpectrum from Exemplar

LeonardoSpectrum is a third party synthesis tool from Exemplar Logic, Inc. LeonardoSpectrum is a synthesis tool that can effectively synthesize VHDL, Verilog, and mixed language designs to create EDIF netlists. ISE 4.x works with LeonardoSpectrum v2000.1b and higher. For general information, see the Exemplar Web site at http://www.exemplar.com/. For detailed information, see the LeonardoSpectrum documentation or the online help provided with LeonardoSpectrum.

Simulation

Simulation verifies the operation of your design before you implement it as hardware. For detailed information about simulation, see the *Simulation and Synthesis Design Guide*.

Simulation Points

Several simulation points are available to test your design:

- Behavioral simulation to check the logic prior to synthesis
- Functional Simulation to check the logic post-synthesis

  ModelSIM simulators are supported in Project Navigator for functional simulation with or without a testbench or test fixture. A testbench or test fixture template generating tool is available in Project Navigator. For automated testbench or test fixture creation, you can use the HDL Bencher.
- Post-MAP simulation to verify behavior post-map
- Post-route Simulation to verify that the design meets the timing requirements you set for your design in the targeted device

  You can perform post-route simulation on your design using ModelSIM (from MTI) and a testbench or test fixture. Post-route simulation allows you to check and correct your design before implementing it. For post-route simulation, you can use the same testbench or test fixture you used for functional simulation. The post-route simulation includes timing information for the targeted device.

Simulation Tools

ISE 4.x supports the following simulation tools:

- HDL Bencher

  HDL Bencher is an automated testbench or test fixture creation tool. The HDL Bencher is fully integrated with Project Navigator.
- ModelSIM Simulator

  ModelSIM from Model Technology, Inc., is integrated in Project Navigator for functional (RTL) simulation of your HDL source modules. ModelSIM XE, the Xilinx Edition of Model Technology,

Inc.'s ModelSIM application, can be installed from the MTI CD included in your ISE 4.x package. For general information, see the Model Technology Web site at http://www.model.com/.

Implementation

The implementation stage consists of taking the synthesized netlists through translation, mapping, and place and route.

To check your design as it is implemented, reports are available for each stage in the implementation process.

Use the Xilinx Constraints Editor to add timing and location constraints for the implementation of your design. You can also open the Xilinx Floorplanner, FPGA Editor, and ChipViewer as necessary.

To check that your design meets timing requirements, Static Timing reports and the Xilinx Timing Analyzer are available.

For more information, see the "Implementation" chapter later in this Guide.

Floorplanner

Floorplanner from Xilinx is an interactive graphical tool that allows you to view and edit location constraints in your design. You can manually or automatically place logic into a floorplan of the selected FPGA. In the Xilinx modular design flow, you can use the Floorplanner to assign location constraints for each module in your design.

The graphical user interface has pull-down menus and toolbar buttons that contain the commands for:

- Changing the design hierarchy
- Floorplanning
- Performing design rule checks

Dialog boxes allow you to quickly set parameters and options for command execution.

For more information, see the "Floorplanner" section of the "FPGA Implementation" chapter.

FPGA Editor

FPGA Editor is a graphical application for displaying and configuring Field Programmable Gate Arrays (FPGAs). Functions you can perform on your designs in the FPGA Editor include:

- Place and route critical components before running the automatic place and route tools.
- Finish placement and routing if the routing program does not completely route your design.
- Add probes to your design to examine the signal states of the targeted device. Probes are used to route the value of internal nets to an IOB for analysis during the debugging of a device.
- Run the BitGen program and download the resulting BIT file to the targeted device.
- View and change the nets connected to the capture units of an ILA core in your design.
- Create an entire design by hand (recommended for advanced users only).

For more information, see the "FPGA Editor" section of the "FPGA Implementation" chapter.

Timing Analyzer

The Timing Analyzer performs static timing analysis of an FPGA or CPLD design. The FPGA design must be mapped and can be partially or completely placed, routed, or both. The CPLD design must be completely placed and routed. A static timing analysis is a point-to-point analysis of a design network. It does not include insertion of stimulus vectors.

The Timing Analyzer verifies that the delay along a given path or paths meets your specified timing requirements. It organizes and displays data that allows you to analyze the critical paths in your circuit, the cycle time of the circuit, the delay along any specified paths, and the paths with the greatest delay. It also provides a quick analysis of the effect of different speed grades on the same design.

The Timing Analyzer works with synchronous systems composed of flip-flops and combinatorial logic. In synchronous design, the Timing Analyzer takes into account all path delays, including clock-to-Q and

setup requirements, while calculating the worst-case timing of the design. However, the Timing Analyzer does not perform setup and hold checks; you must use a simulation tool to perform these checks.

The Timing Analyzer creates timing analysis reports, which you customize by applying filters with the Tab dialog box options.

For more information, see the "Timing Analyzer" section of the "FPGA Implementation" chapter.

XPower

XPower is a post-design application tool that allows you to interactively and automatically analyze power consumption for the following Xilinx FPGAs and CoolRunner CPLDs:

- Virtex
- VirtexE
- Virtex II
- Spartan II
- Spartan E
- XPLA3

XPower does *not* support:

- Coolrunner2
- XC4000
- XC5000
- XC9000
- Virtex2Pro

For more information, see the "XPower" section of the "FPGA Implementation" chapter.

ChipViewer

The ChipViewer provides a graphical view of the fitting report. You can examine inputs and outputs, macrocell details, equations, and pin assignments. You can examine both pre-fitting and post-fitting results. The ChipViewer also accesses the Timing Analyzer Report.

- Pre-fitting Examination

  The pre-fitting examination looks at the contents of the .ngd file, the design netlist. In this view you can get basic I/O information and make pin assignments, pin-lock, modify your UCF file and prohibit pins.

- Post-Fitting Examination

  The post-fitting examination looks at the contents of the .vm6 file, the fitting result. Here you can examine final results.

For more information, see the "CPLD ChipViewer" section of the "CPLD Implementation" chapter.

Device Programming

When your design meets all your requirements, you can create a programming file that can be downloaded to the target device. For more information, see the "Device Programming" chapter later in this Guide.

Use the following tools to program a device:

- "iMPACT"
- "PROM File Formatter"

iMPACT and PROM File Formatter can be opened from Project Navigator.

iMPACT

iMPACT, a command line and GUI based tool, allows you to:

- Configure your PLD designs using Boundary-Scan, Slave Serial, and Select Map configuration modes.
- Download.
- Read Back and Verify design configuration data.
- Perform functional tests on any device.

For more information, see the "iMPACT" section of the "Device Programming" chapter.

PROM File Formatter

Use PROM File Formatter to:

- Format BIT files into a PROM file that is compatible with Xilinx and third-party PROM programmers.
- Concatenate multiple bitstreams into a single PROM file for daisy chain applications.
- Store several applications in the same PROM file (using the Xilinx FPGA reconfiguration capability).

For more information, see the "PROM File Formatter" section of the "Device Programming" chapter.

Chapter 5

HDL

This chapter contains the following sections:

HDL Sources

This section describes the creation of HDL design sources. You can create your design using:

- HDL code only

  OR

- A combination of HDL code, schematics, and state diagrams

Supported Languages

ISE 4.x supports the following languages for the creation of HDL source files:

- "VHDL"
- "Verilog"
- "ABEL-HDL"

Note ABEL-HDL is supported for CPLD designs only.

The languages that can be included in your project depend on the targeted device and design flow.

VHDL

VHDL is an industry-standard hardware description language. It is recognizable as a file with a .vhd or .vhdl extension. VHDL stands for ***V***HSIC (Very High-Speed Integrated Circuits) ***H***ardware ***D***escription ***L***anguage.

You can use VHDL to model a digital system at many levels of abstraction, ranging from the algorithmic level to the gate level. VHDL is capable of describing the concurrent and sequential behavior of a digital system with or without timing.

See the *Xilinx Synthesis Technology (XST) User Guide* for information on using VHDL in projects with the XST synthesis tool. See the *Synthesis and Simulation Design Guide* for information on using VHDL in projects with the FPGA Express synthesis tool and other tools.

Verilog

Verilog is a commonly used Hardware Description Language (HDL). It can be used to model a digital system at many levels of abstraction, ranging from the algorithmic level to the gate level. Verilog files have a .v extension.

Originally developed by Cadence Design Systems, Verilog is now maintained by OVI (Open Verilog International). For more information, go to http://www.verilog.com.

See the *Xilinx Synthesis Technology (XST) User Guide* for information on using Verilog in projects with the XST synthesis tool. See the *Synthesis and Simulation Design Guide* for information on using Verilog in projects with the FPGA Express synthesis tool and other tools.

ABEL-HDL

ABEL is a high-level language (HDL) and compilation system. In ISE 4.x, ABEL-HDL is supported for CPLD devices only. It is not supported for FPGA devices or for CPLDs devices used with a design flow other than ABEL XST or ABEL BLIF.

You can convert existing ABEL-HDL designs into VHDL or Verilog design for use with other devices or synthesis tools. To access an HDL Converter:

1. Click the Device and Design Flow line in the Sources in Project window.
2. Right-click HDL Converter (under Design Utilities) in the Processes for Current Source window, and select Properties.
3. In the Process Properties dialog box, enter the name of the ABEL file you want to convert.
4. Choose whether to convert it to VHDL or Verilog.

Creating HDL Source Files

To create an HDL source file:

- Use any text editor to create an HDL source file, then add that file to your project.

  OR

- Use Project Navigator's New Source Wizard and HDL Editor.

New Source Wizard

The New Source Wizard displays a series of dialog boxes in which you enter information about the new source. Project Navigator uses this information to open a skeleton file in HDL Editor using the selected language. You choose whether you want to add the file to the project.

Creating New HDL Modules

The following procedure describes the creation of a VHDL file. The procedure for creating Verilog or ABEL-HDL files is similar.

To create a new HDL Module:

1. Open or create your project (see the "Creating a Project" section of the "Projects" chapter).
2. Click **`Project`** → **`New Source`** from Project Navigator to access the New Sources in Project window.

3. Select the HDL file type you want to create from the list of available source types displayed in the New window. The HDL source types included in the list depend on the device and design flow you selected for your project.
4. Enter a name for the new HDL file in the File Name box. See the "Creating a New Source" section of the "Projects" chapter for detailed information on the New source dialog box.
5. Click **Next**.

   For new VHDL modules, the Define VHDL Sources in Project window opens. Use this window to create skeleton code for the VHDL module you are describing.
6. Enter a Port Name.
7. Click in the right side of the Direction box.
8. In the drop-down list, select a direction (in, out, inout).
9. Click in the right side of the Most Significant Bit (MSB) box and Least Significant Bit (LSB) box to access their selector arrows. Use the up and down arrows to select the value.

   Note The MSB and LSB fields define the signal on the pin name as a bus. For example, for a pin named DATA with an MSB of 7 and an LSB of 0, the bus would be DATA[7:0]. If the signal on the pin is not a bus (for example, a clock), leave the MSB and LSB fields blank.
10. Click **Next**.

    The New Source Information window opens with a summary of the specifications made in the Define VHDL Sources in Project window.
11. Click **Finish**.

    The HDL Editor opens in the Project Navigator's workspace with the newly created skeleton code displayed in it.
12. Use the HDL Editor to continue coding from the new module.

Creating a New VHDL Package

The following procedure describes the creation of skeleton code for a new VHDL package. The procedure to create skeleton code for a new VHDL testbench or Verilog test fixture is similar.

To create skeleton code for a new VHDL package:

1. Open or create your project (see the "Creating a Project" section of the "Projects" chapter).
2. Click **Project** → **New Source** from Project Navigator to open the New Sources in Project dialog box.
3. Click **VHDL Package** from the list of available source types displayed in the New dialog box.
4. Enter a name for the new VHDL package in the File Name box. See the "Creating a New Source" section of the "Projects" chapter for detailed information on the New source dialog box.
5. Click **Next**.

   The New Source Information window opens with a summary of your request.
6. Click **Finish**.

   The HDL Editor opens with skeleton code for a new VHDL package.
7. Use the HDL Editor to complete the VHDL package.

Opening HDL Source Files

To open any HDL source file listed in the Sources in Project window:

1. Double-click the HDL file name.
2. The HDL file opens in the HDL Editor window of Project Navigator.

The HDL Editor is language sensitive and identifies the language in the file by the file extension.

HDL Editor

The HDL Editor is a text editor for editing HDL source files. In addition to regular editing features, the editor provides syntax coloring. The syntax coloring feature supports three languages:

- VHDL
- Verilog
- ABEL

To open HDL Editor:

- Click **Project** → **New Source** from the Project Navigator. Follow the new source creation sequence. At the end of the sequence, the new HDL file opens in the HDL Editor.

  OR

- Double-click any HDL file listed in the Sources in Project window.

  OR

- Click **File** → **New** from the Project Navigator.

  Note **File** → **New** does not add the new file to the project. You must use **Project** → **Add Source** if you use **File** → **New** to create a new HDL source.

HDL Editor Functions

HDL Editor Online Help

Detailed procedures for using the HDL editor are given in the HDL Editor online help.

General Help

To access comprehensive HDL Editor online help:

1. Click **Help** → **ISE Help Contents** in the Project Navigator menu.

   The Xilinx ISE Online Help System menu (the help umbrella) opens.

2. Click **HDL Editor** under Design Entry.

   The HDL Editor help window opens.

3. Search for your topic using the Contents, Index, or Find tabs.

Context Sensitive Help

Context sensitive online help, especially for reserved words, is available in open HDL files. To access context sensitive online help:

1. Highlight a word or phrase.
2. Press **F1**.
3. A context sensitive help topic is displayed.

File Operations

Use the Project Navigator **File** menu to open, close, print, and save files in the HDL Editor workspace. You can have multiple files open at one time. Use the tabs at the bottom of the HDL Editor window to move between files.

Window Operations

To maximize the HDL Editor workspace:

- Click the Toggle Workspace Window toolbar button to hide the Sources in Project and Processes for Current Source windows.

- Click the Tool Transcript View toolbar button to hide the Transcript view.

See the “Docking and Undocking Windows and Toolbars” section of the “Project Navigator” chapter for additional information on maximizing the HDL Editor workspace.

Editing Functions

The Project Navigator Edit menu contains the cursor movement, selection, copy, cut, paste, and insert file functions that you use with the HDL Editor. Icons on the Project Navigator Toolbar are also available for the basic functions.

When the Language Template window is active, the Project Navigator Edit menu changes to reflect the edit functions available with the Language Templates tool. A separate toolbar on the Language Template window is also available for these functions.

Note For information about Language Templates, see the "Language Templates" section below.

Search Functions

The Project Navigator Edit menu includes search functions for the files in the HDL Editor workspace. The Find in File function is particularly useful for searching across multiple files. You can initiate the search from the Project Navigator Edit menu, or from the search input field of the Editor toolbar. The Find in Files function searches all files currently open in the HDL Editor workspace for the specified target. The results appear in the Transcript window. The input field on the Editor toolbar includes a list of previous search targets.

Macro Functions

The Project Navigator Macro menu contains functions for use in keyboard recording and playback.

The HDL Editor allows you to record your own macros for use in the active document. You can file macros in user created directories and recall them in future documents.

Note You are not prompted to save your macros when closing the Project Navigator. Select Save Macros to make your recorded macros available for future projects.

Customizing Tabs and Fonts

To customize the tab settings and fonts used with the HDL Editor:

1. Click **Edit** → **Preferences** from the Project Navigator.
2. Click the **Editor** tab on the Preferences dialog box.
3. Use the Tabs and Font areas on the Editor Preferences menu to modify these items.

Note The background color of the HDL Editor workspace is controlled by your computer's application background display setting.

Language Specific Features

The HDL Editor identifies the coding language in a file based on the extension added to the file name. See Table 3-1 of the "Projects" chapter for source type and file extension information.

The HDL Editor includes color coding of strings, comment, keywords, and directives as well as context sensitive help for reserved words. The HDL editor adjusts its color-coding, help, and keyword and reserved word identification as appropriate for the language contained in the file. To check whether a word in an open HDL file is a reserved word, highlight it and press **F1.** This opens the HDL Editor help contents, which contains information on reserved words.

A Language Templates tool is included to aid VHDL, Verilog and ABEL source code entry. See the "HDL Sources" section above for information on this tool.

Language Templates

Multiple language and synthesis templates with prepared pieces of code are available in ISE 4.x. These templates enable easy insertion of pre-built text structures such as common language structures or instantiation templates for synthesis into your HDL source file. There are four types of templates:

- Component Instantiation
  For VHDL and Verilog only

- Language Templates
  With basic language constructs
- Synthesis Templates
  With synthesis-oriented implementation of basic functional blocks, such as multiplexers, flip-flops, and counters
- User Templates
  User created templates for specific constructs

Opening the Language Templates Tool

To open the Language Templates tool:

- Click **`Edit`** → **`Language Templates`** from the Project Navigator.

  OR

- Click the Language Template icon in the Editor toolbar.

Figure 5-1 Language Template Icon

The Language Template window opens in the HDL Editor window. See the following figure.

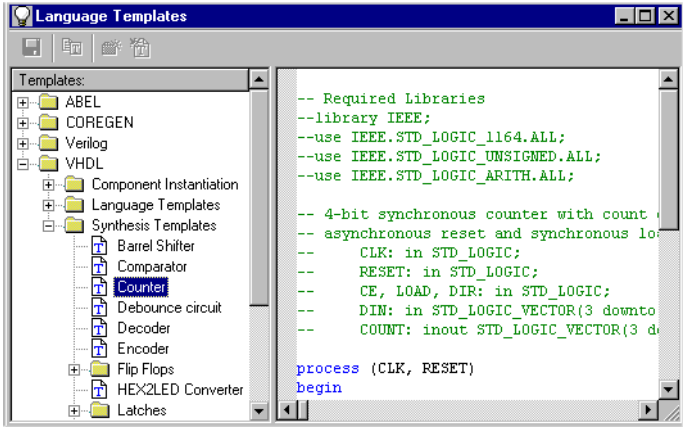

Figure 5-2 Language Templates Window

The Language Templates window consists of two panes:

- The left pane allows you to list the available code templates for ABEL, Verilog, VHDL, and the CORE Generator components.
- The right pane displays the template information (text) when you select a template.

Selecting an Existing Template

To select a template for use in your HDL code:

1. Open the Language Templates window as described in the "Opening the Language Templates Tool" section.
2. Click the "+" symbol in front of **ABEL**, **COREGEN**, **Verilog**, or **VHDL** to display the directories under each selection (see the figure in the "Language Templates" section).
3. Click the directory under the selected language or COREGEN that represents the type of template you want. Browse through the underlying directories and files to select the template.
4. Click a template name to display the template (and example code for Language Templates) in the right-hand pane.

You cannot edit component instantiation templates, language templates, or synthesis templates from the Language Templates window. You must copy them to an open file in the HDL Editor workspace to modify template contents. After you modify the contents or create a new template, you can add the template as a user template for future use.

Inserting Templates in HDL Sources

Use any of the following methods to insert a template into an open source file in the HDL Editor workspace.

Method One

1. Select a template from the left-hand pane of the Language Templates window.
2. Drag and drop it into the HDL code.

   To scroll within the HDL Editor to the correct line, continue to hold the left mouse button down, and move to the top or bottom

of the Edit window. A special cursor indicates where the text will be dropped.

Method Two

1. Place your cursor where you want to insert the text.
1. Highlight the template text.
2. Click **Edit** → **Use in File** in the Project Navigator, or click the Use in File icon on the Language Templates toolbar.
3. The Template text is inserted into the HDL file.

Method Three

1. Highlight the template text.
2. Right-click and select copy.
3. Place your cursor where you want to insert the text.
4. Right-click.
5. Click **Paste** from the pull-down menu.

Method Four

1. Highlight the text you want to insert.
2. Drag-and-drop it into your document.

Creating a User Template

To create a new user template:

1. Open the Language Templates window as described in the "Opening the Language Templates Tool" section.
2. Select **ABEL**, **Verilog**, or **VHDL** from the Templates window of the Template Assistant.
3. Highlight the **User Templates** directory under the selected language.
4. Click the **New Template** icon on the Template Assistant toolbar.
5. Enter a name for the new template in the New Template name edit field that appears in the Language Templates window.

6. Move the cursor to the window on the right side of the Template Assistant and use the HDL Editor to enter the template text.
7. Click the Save Templates icon on the Template Assistant toolbar when you are ready to save the newly entered template information.

Use the methods described in the "Inserting Templates in HDL Sources" section to place the template in your HDL code.

Creating a Schematic Symbol from an HDL Source

To create a schematic symbol from an HDL file for use on a schematic:

1. Highlight the HDL file in the Sources in Project window. The file can contain underlying schematics or other HDL modules.
2. Double-click the **Create Schematic Symbol** process in the Processes for Current Source window.

The symbol is created and placed in the project directory. It is named the same as the HDL file except that it has .sym as its extension. It is also automatically included in the Local Symbol directory of the Engineering Capture System (ECS) symbol libraries.

To use the symbol in a schematic:

1. Open a schematic.
2. Click **Add** → **Symbol** in the Engineering Capture System (ECS).
3. Select the symbol from the Local Symbol directory for placement on the schematic.

   See the "Schematic Sources" chapter for information on the Engineering Capture System (ECS).

To view and modify the HDL text after you add a symbol created from a HDL source:

1. Click **View** → **Push/Pop** from the Engineering Capture System (ECS) menu.
2. Select the HDL source symbol on the schematic.

   The HDL source file opens in the HDL Editor workspace.

HDL Library Mapping

ISE 4.x includes an HDL library mapping feature to include HDL files as part of an HDL library. Libraries allow sharing of design elements between sources in a design and between designs. This feature makes designs easier to manage and allows for design reuse. The HDL library mapping feature allows you to identify library files, and depending on the language, the named library they are part of. This information is passed to the various tools that need it, such as synthesis tools and simulators.

VHDL

VHDL requires all design sources to be in a library. VHDL also allows named libraries that can contain one or more files. VHDL design units can access other design units in the same and different libraries by declaring the name of the library and design unit to make visible.

For example, take an entity called *foo* that wants to access a function called *foofunc* that is declared in a package named *foopack* that is in a library named *foolib*. The following declaration for *foo* allows you to access everything declared in the package *foopack*.

```
library foolib;
use foolib.foopack.all;
entity foo is
...
end entity;
```

In all VHDL tools there is a compile order dependency between libraries. Libraries must be created and compiled before design units that use them can be compiled. In the above example, the library *foolib* must be created and the package *foopack* compiled into it before the entity *foo* can be compiled.

Currently you must compile the libraries manually for simulation and synthesis.

Verilog

Verilog does not support named libraries. All modules are global and are visible to all other modules. However, most Verilog tools support the idea of a library as either a directory of files that each contain a single module, or single files that contain many modules. In either

case, when a tool is compiling the design and encounters a module instantiation that is not resolved in the set of input files, the tool looks in the specified library files or in the library directories to find the module definition.

There is a distinction between design files that are part of a project and library files. Design files are always compiled completely by the simulator or synthesis tool. Libraries are compiled as they are needed, and then only the required module is compiled. For example, if the synthesis tool needs a module called *foomod* from a library file that contains many modules, it compiles and uses *foomod* only, and ignores the rest of the modules.

Project Navigator Source Libraries

In ISE 4.x, the default library in which the current design sources are placed is named *Work*. Depending on the synthesis tool for the project, a VHDL or Verilog library is also provided. The libraries and elements included in these libraries are listed in the Library View of the Sources in Project window.

The Work library contains all the design sources in the project. It corresponds to the sources in the Module View. The Work library cannot be deleted. Files in the work directory cannot be in other libraries at the same time. When you use the New Source, Add Source, or Add Copy of Source options in the Project menu, the files created or added through these options appear in the Work library but can be moved to other libraries. See the "Moving Files to a Library" section below for more information.

If you are using FPGA Express, XST VHDL, or ABEL XST, the VHDL library is included in the Library View. The VHDL library cannot be deleted. The VHDL library contains named VHDL libraries. You can create named libraries within the VHDL library as described in this section and add any number of files to those libraries.

If you are using FPGA Express or XST Verilog, the Verilog library is included in the Library View. The Verilog library contains Verilog files. The Verilog library cannot be deleted.

Named VHDL Libraries

You cannot directly add a file to the VHDL library displayed in the Library View. You must first create a named directory to hold the file.

To create named VHDL directories:

1. Open or create your project (see the "Creating a Project" section of the "Projects" chapter).
2. Click **Project** → **New Source** from the Project Navigator.
3. Select **VHDL Library** from the list of available source types displayed in the New dialog box. The VHDL Library selection is available only with projects that use the XST VHDL, ABEL XST, or FPGA Express synthesis tools.
4. Enter a name for the new VHDL library in the File Name box.
5. Click **Next**.
6. Click **Finish** in the New Source Information dialog box.
7. Click the Library View tab in the Sources in Project window. The newly created VHDL library appears under VHDL.

Adding a File to the Library

To add a file to the library:

1. Right-click **library** in the Library View tab.
2. Select **Add Source**.
3. Choose the source you want to add to the library.

See the "HDL Library Mapping" section for more information.

Renaming VHDL Libraries

To rename a VHDL library:

1. Go to Library View in the Sources in Project window.
2. Click the name of the VHDL library you want to rename.
3. Click **Source** → **Rename** from the Project Navigator menu.
4. Modify the library name.

Removing VHDL Libraries

To remove a VHDL library:

1. Go to Library View in the Sources in Project window.
2. Click the name of the VHDL library you want to remove.
3. Click **Source** → **Remove** from the Project Navigator menu.
4. A message box reminds you that removing a library may cause the implementation data to be out of sync. Click **Yes** to remove the library and all of its files from the project.

   Note The library is deleted. Its files are not deleted, just removed from the project.
5. Click **Project** → **Delete Implementation Data** after you remove the library.

   Caution After a library has been deleted there is no way to restore it except to recreate it.

Moving Files to a Library

Files that have been added to the project can be moved from one library to another. A file cannot be moved to a library that does not support its file type. For example, you cannot move a Verilog file to a VHDL library. Any file may be moved to the Work library.

To move files between libraries:

1. Go to the Library View tab.
2. Click the name of the file you want to move.
3. Click **Source** → **Move to Library**.
4. Select the library from the Choose Library dialog box.

The selected file is moved to the chosen library. The Library View reflects the move.

Removing Files from a Library

When you click a file name in any library in the Library View and select **Source** → **Remove**, the file is removed from the library and the project. The file is not deleted.

Chapter 6

State Diagrams

This chapter contains the following sections:

About StateCAD and StateBench

ISE 4.x includes integrated support for StateCAD® and StateBench™ for state machine design entry and verification

StateCAD automates the creation and development of state machines and their translation to HDL code. StateCAD includes:

- A State Machine Wizard to help you develop the initial state machine
- A Logic Wizard to create data flow structures
- An Optimization Wizard to maximize performance for the target device

StateCAD also identifies many kinds of design problems for you. When your design is error free, StateCAD translates it into synthesizable VHDL, Verilog, or ABEL-HDL code that can be used in your ISE 4.x project.

StateBench provides behavioral verification of StateCAD state diagrams.

Creating a New State Diagram

To create a new state diagram source:

1. Click **Project** → **New Source** from the Project Navigator.
2. Click **State Diagram** from the list of sources displayed in the New source dialog box.
3. Enter a name for the state diagram in the File Name field.

   A StateCAD state diagram name has the following requirements:

   - The name must begin with an alphabetic character.
   - StateCAD is not case sensitive. File names are recognized independent of case.
   - The name can contain up to eight alphanumeric characters. It must follow the DOS 8.3 convention. However, do not add the three character extension. The appropriate extension (.dia) is automatically generated.
   - The name defaults to *untitled.dia* in StateCAD if the above rules are not followed.

   Note The name entered here is used by StateCAD in the module or entity definition in the HDL code it creates from the state diagram.
4. By default, StateCAD saves the new source in the project directory. Specify a different directory in the Location field on the New window if desired.

   Note The Add to Project check box on the New window has no effect on state diagrams. State Diagrams cannot be automatically added to a ISE 4.x project.
5. Click **Next**.

   The New Source Information dialog box opens, summarizing the requested information for the new source.
6. Click **Finish** on the New Source Information dialog box to open StateCAD.

   The StateCAD main window opens with the newly specified state diagram loaded, ready for you to begin designing the state machine.

7. Create the state diagram. The following references and resources are available:

   - StateCAD online help

     For help using StateCAD, see the StateCAD online help.

   - StateCAD tutorial

     A StateCAD tutorial is available from the online help. Click **Help** → **Tutorial**.

   - Design Wizards

     Click **File** → **Design Wizard** from the StateCAD menu to initiate the Design Wizard. A dialog box opens asking you to verify whether you want to use the currently opened state diagram or a new one. Click **Yes**.

8. The StateCAD Design Wizard leads you through the design process. Project information (such as synthesis tool, targeted device) is reflected in the Design Wizard dialog boxes.
9. Click **Save File** to save your state diagram.
10. Click the Generate HDL button to generate HDL for your state diagram.
11. Close StateCAD.
12. Add the state diagram .dia file and the generated HDL module to your ISE project by clicking **Project** → **Add Source**.

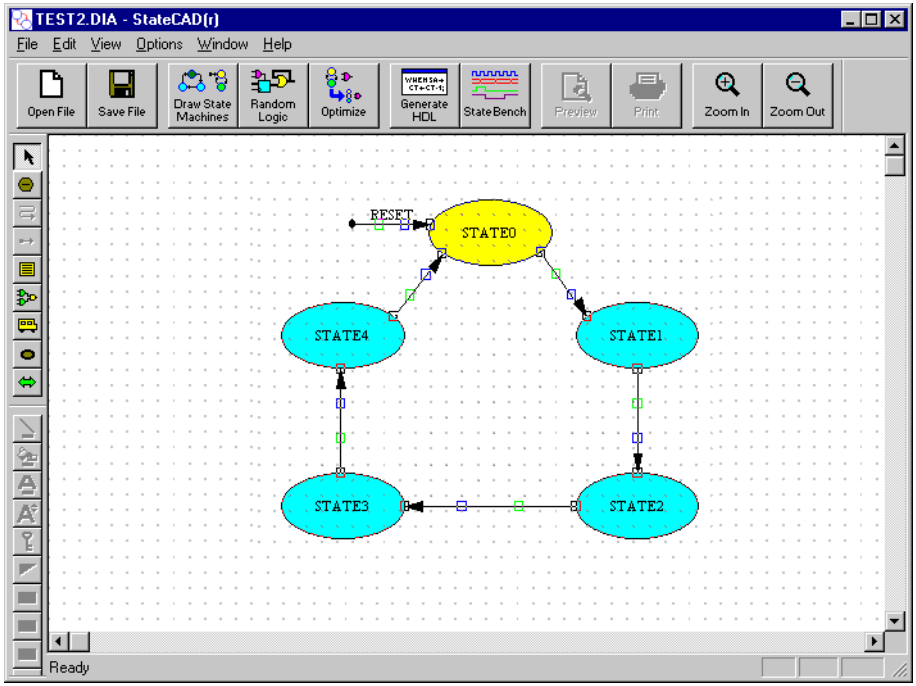

Figure 6-1 StateCAD Main Window with Diagram Loaded

Updating an Existing State Diagram

To open existing StateCAD state diagrams (.dia files) in StateCAD from the Project Navigator:

1. Double-click the state diagram name in the Sources in Project window.
2. The state diagram is opened in StateCAD.

When you modify and save a valid state diagram, StateCAD prompts you that the HDL file needs to be updated. If you select to update the HDL file, the HDL file is regenerated automatically based on the entered changes. If the corresponding HDL source file has been added, it is automatically overwritten with the updated version.

Using StateBench

To open StateBench from StateCAD, click **Options** → **StateBench (Create Testbench)**. Use StateBench to verify the StateCAD design's behavior and validate its timing.

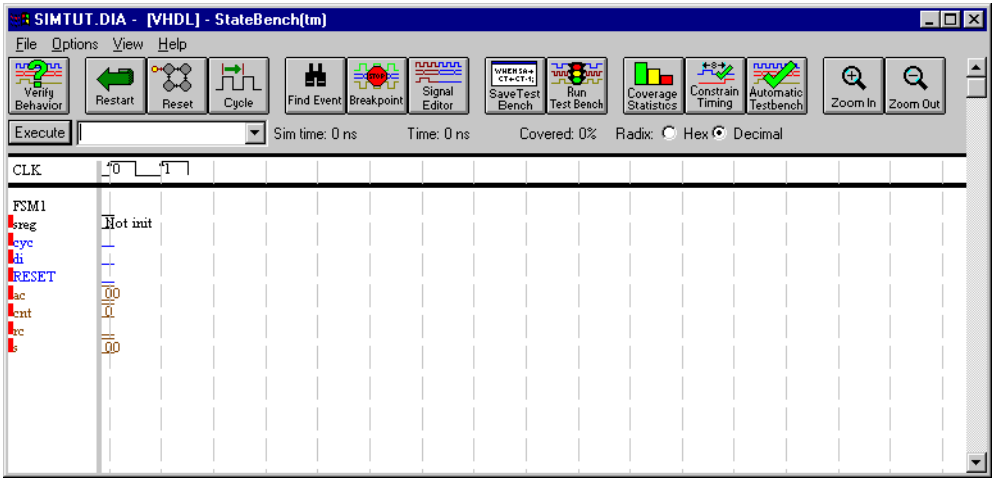

Figure 6-2 StateBench Diagram

You can also add and use StateBench testbenches. For help using StateBench, see the StateBench online help. A StateBench tutorial is also available from the online help (**Help** → **Tutorial**).

Instantiating State Diagram Modules

For information on instantiating state diagram modules in Schematic designs, see "Creating the State Machine Macro" section of the ISE In-Depth tutorial, available from the Xilinx support Web site at http://support.xilinx.com/support/techsup/tutorials/. Instantiating state diagram modules in HLD designs is the same as instantiating any other HDL module.

Chapter 7

Schematic Sources

This chapter contains the following sections:

Schematic Source Files

This chapter:

- Describes how to use schematic sources in ISE 4.x projects.
- Discusses the basic concepts for using the Engineering Capture System (ECS) to edit schematics and symbols.

Your ISE 4.x project can include schematics as well as HDL sources. You initiate the creation of a schematic source from the Project Navigator. After the source file is created, Project Navigator opens the Engineering Capture System (ECS) for you to create and modify the schematic design.

There are two principal components to ECS:

- ECS Window

  The ECS window is the main interface for creating schematics and symbols.

- HDL netlisters

  The netlister programs translate a schematic into the HDL model used for design synthesis, simulation, and implementation of the design.

This section contains information on the Project Navigator's interaction with schematic source files. See the "ECS (Engineering Capture System)" section for information on using ECS to edit schematics and symbols. The most detailed information on using the ECS tools is in the ECS online help (from the **Help** menu and **F1** context sensitive help).

Creating a Schematic Source File

To create and add a schematic source (.sch):

1. Open or create your project as described in the "Creating a Project" section of the "Projects" chapter.
2. Click **Project** → **New Source.**
3. In the New dialog box, click **Schematic**.
4. Enter a name for the new schematic in the File Name field. The project location is automatically entered in the Location field.

   Note Check the Add to project box to automatically add the schematic file.

5. Click **Next**.
6. In the New Source Information dialog box, click **Finish**.
7. The Project Navigator opens ECS (Engineering Capture System). See the following figure. ECS opens with a new schematic sheet for the newly created schematic source file. The schematic source file containing the schematic is named as specified in the New source dialog box plus an .sch extension.

If you chose to add this file to the project, the new schematic file is automatically added and listed in the Project Navigator Sources in Project window.

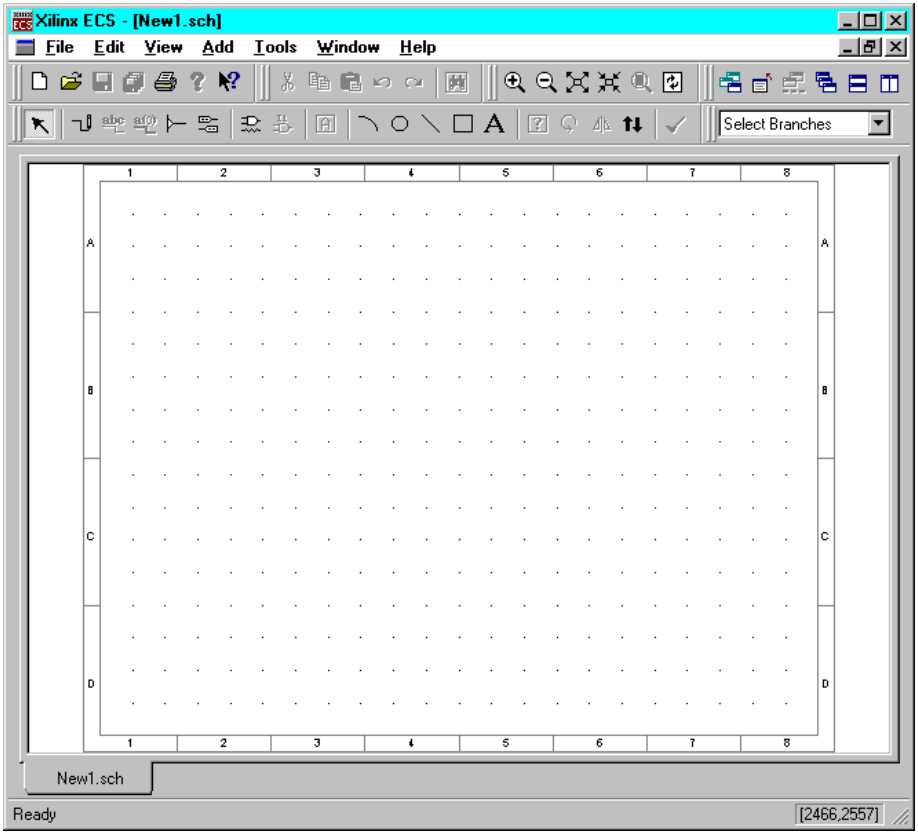

Figure 7-1 ECS Window

You can now use ECS to create the schematic module. See the online help for details on using ECS. See the "Guidelines for Creating Schematics" section below for important information on creating schematics for use with the synthesis tools.

Opening a Schematic Source File

To open an existing schematic source (.sch):

1. In Project Navigator, go to the Sources in Project window.
2. Double-click the schematic source name.

   ECS opens with the schematic you selected.

Updating Schematic Files

The schematic file database tracks dates of symbol files used in schematics. Schematics can become out of date when symbols are edited or updated (newer dates than those saved in the schematic). After modifying symbols used in your schematic source files, run **`Update all Schematic Files`** within Project Navigator to update all the schematics in your design with the current symbols in your symbol directories.

To update the schematic files:

1. Click the Device and Design Flow line in the Sources in Project window.
2. Double-click **`Update all Schematic Files`** in the Processes for Current Source window.

If you open a schematic in ECS with out-of-date symbols, a dialog box opens to let you update these symbols within ECS. To update all of the schematic files, you run the update from Project Navigator.

Xilinx Implementation Attributes and Constraints

It is possible to use the attribute function in ECS to pass constraint information from your schematic to the Xilinx implementation tools. See the ECS online help for detailed information on ECS schematic attributes.

Certain Xilinx implementation attributes, such as the INIT attribute, must be set on the component in the schematic for correct simulation results. The basic procedure to set an INIT attribute on a block RAM component (Virtex), for example, in an ECS schematic is:

1. Create a new schematic in ECS.
2. Add the block RAM components.

3. Select a block RAM symbol in the schematic.
4. Click **Edit** → **Object Properties**.
5. In the Object Properties dialog box, select the **INIT** attribute you want to set.
6. Click New.
7. In the New Attribute dialog box, enter the correct INIT string in the **Attribute Value** field.
8. Click **OK** to close the New Attributes dialog box.
9. Click **OK** to close the Object Properties dialog box.

   The INIT value is set for the selected schematic component.

Instantiating HDL Sources

You can instantiate HDL code sources into a schematic by first creating a schematic symbol for the HDL source. The HDL sources must be in the project directory.

Creating a Schematic Symbol

To create a schematic symbol for an HDL source:

1. In Project Navigator, click an HDL source file in the Sources in Project window.
2. Double-click **Create Schematic Symbol** in the Processes for Current Source window.

The symbol (.sym) is automatically added to the Local symbol library and is available for use in ECS.

Symbol Generator Options

By default, the Create Schematic Symbol process does not automatically overwrite an existing symbol of the same name as the symbol being generated. A process property allows you to specify whether or not to overwrite the existing symbol.

To set this property:

1. Click an HDL source file in the Sources in Project window.
2. Right-click **Create Schematic Symbol** in the Processes for Current Source window.
3. Click **Properties**.
4. Click in the Value box for the **Overwrite Existing Symbol** option to toggle this property on and off.

Opening the HDL Source

If you have instantiated HDL sources in a schematic, you can access the HDL sources from ECS.

To access HDL sources:

1. Open the ECS window.
2. Select a symbol representing an instantiated HDL source in the schematic.
3. Click **View** → **Push Into Symbol or Return to Calling Schematic**.

   The HDL source file opens in the HDL Editor window of the Project Navigator.

If you modify and save the HDL source in the HDL Editor, you must click **Create Schematic Symbol** in the Project Navigator Processes for Current Source window to replace the symbol used in the schematic with the updated source. If the ports change, you must also reconnect the updated symbol to the schematic.

Creating a Top-Level Schematic

To use schematics in your ISE 4.x project, create a top-level schematic and instantiate your HDL design sources into the schematic as follows:

1. Create the HDL design sources and add them to the project.
2. To create a symbol for each HDL source, click the source in the Sources in Project window, then click **Create Schematic Symbol** in the Processes for Current Source window. The symbol

(.sym) is automatically added to the Local symbol library used in ECS.

3. To create a schematic source (.sch), click **Project** → **New Source**, then select Schematic as the source type.
4. In ECS, click **Add** → **Symbol**. Select each of the symbols created for your HDL sources from the symbol browser and place them on the sheet.
5. Click **Add** → **Wire** and connect the HDL source symbols as appropriate. Save the schematic (.sch) and exit ECS when you are finished.
6. The Project Navigator automatically recognizes the design hierarchy and moves the top-level schematic source (.sch) to the top of the Sources in Project window's design tree with the HDL sources listed under it.

Note See the *ISE Tutorial* for an example. See the ECS online help for instructions on adding wires, naming nets and buses, and adding I/O markers and other items to complete the top-level schematic.

Simulating and Synthesizing Schematic Sources

An HDL netlist is created from all schematic sources and used for simulation and synthesis.

- For projects that use the XST VHDL or FPGA Express VHDL synthesis tools, the VHDL netlister program automatically generates a VHDL functional model for any schematic source in a project. See the "VHDL Functional Model" section for more information on the generated model.
- For projects that use the XST Verilog and FPGA Express Verilog synthesis tools, the Verilog netlister program generates a Verilog Functional Model for schematics. See the "Verilog Functional Model" section for more information on the generated netlist.

Simulation of schematic sources requires a testbench (VHDL) or test fixture (Verilog). Use the HDL Bencher and the netlist generated from the schematic to create the testbench or test fixture. See the *Synthesis and Simulation Guide* for information on simulating and synthesizing designs.

VHDL Functional Model

The VHDL netlister program automatically generates a VHDL model for any schematic source in a project when a VHDL simulation process is run. The VHDL model consists of an entity declaration and an architecture.

The VHDL netlister uses the following conventions when generating the VHDL functional model for a schematic:

- The name of the schematic becomes the name of the entity.
- Each net name flagged with an I/O marker is declared as a port in the entity declaration.
- The architecture name is always *schematic*.
- Scalar nets become VHDL signals of type *std_logic*.
- Buses become VHDL signals of type *std_logic_vector*.
- Component declarations are generated in the architecture for each type of symbol instantiated in the schematic.
- A component instance statement is created for each symbol instance on the schematic. The symbol instance name becomes the statement label.
- Each symbol pin becomes a port on the corresponding component.

Viewing the VHDL Functional Model

After you create the schematic in ECS, the Project Navigator uses the VHDL functional models for all further processing of the design.

To view the VHDL functional model for a schematic:

1. Click the schematic in the Sources in Project window.
2. Double-click **`View VHDL Functional Model`** in the Processes for Current Source window.
3. The VHDL model displays in the ISE Report Viewer.

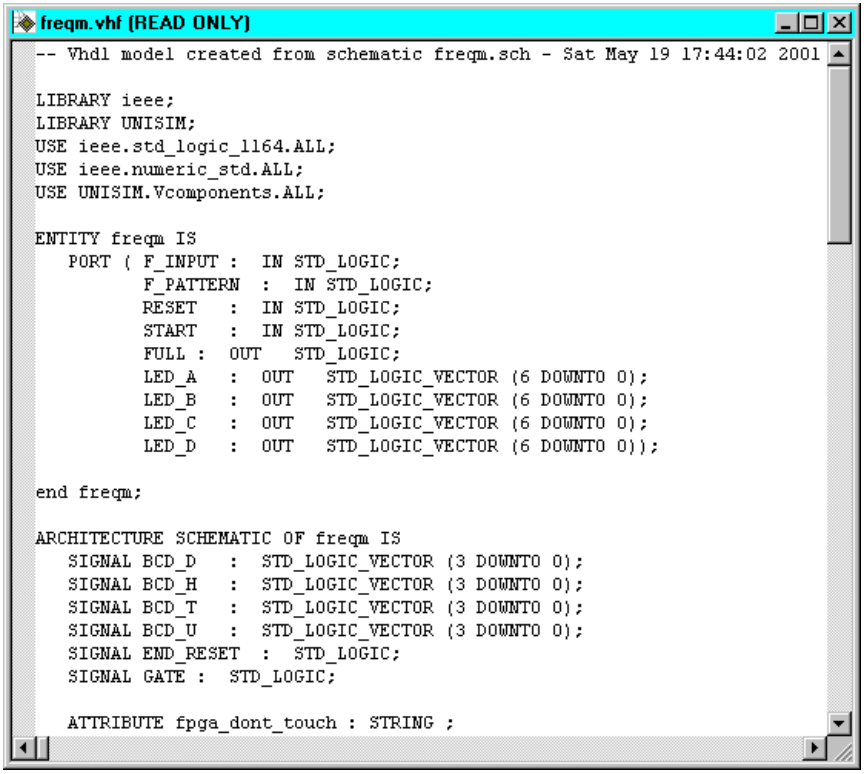

Figure 7-2 VHDL Functional Model Example

Verilog Functional Model

The Verilog Netlister program automatically generates a Verilog Functional Model for any schematic source in a project. After you create the schematic in ECS, the Project Navigator uses the Verilog Functional Model for all further processing of the design.

To view the Verilog netlist for a schematic:

1. Click the schematic in the Sources in Project window.
2. Double-click **`View Verilog Functional Model`** in the Processes for Current Source window.
3. The Verilog netlist displays in the report window.

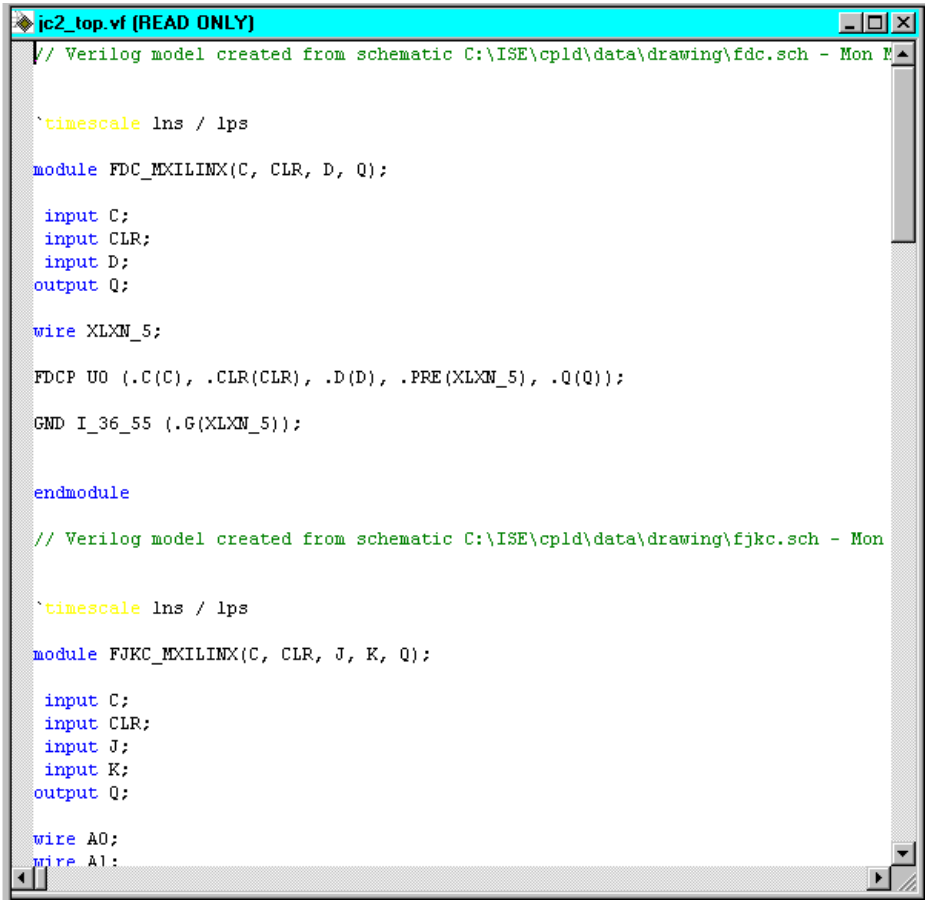

Figure 7-3 Verilog Netlist Example

ECS (Engineering Capture System)

ECS (Engineering Capture System) is the design entry and analysis tool for schematic sources in ISE 4.x projects. It includes the following features:

- Schematic Capture

  ECS captures your design logic in schematic form.

- Libraries

  Xilinx device-specific symbol libraries are provided for schematic creation. You can create local symbols in a symbol window as needed, or you can access ECS symbols from other projects.

- Symbol creation

  ECS can open a symbol window in which you can create your own symbols and give them whatever characteristics you want. Alternatively, you can convert a schematic or HDL source file into a Block symbol to make your design easier to understand or for reuse in other projects.

- Schematic and Symbol Check

  You can check your schematics and symbols at any time for such errors as unconnected wires or illegal pin names.

- Netlist Generation

  For use in ISE 4.x project, all schematics are converted into a VHDL or Verilog netlist depending on the synthesis tool you selected for your project (see the "Selecting a Device and Design Flow" section of the "Projects" chapter). When you select a schematic source in the Sources in Project window, click the **`View VHDL Functional Model`** or the **`View Verilog Functional Model`** process in the Processes for Current Source window to see how the schematic was converted.

- Simulator Interface

  You can use testbenches and the ModelSIM simulators to verify schematic designs. A testbench (VHDL) or test fixture (Verilog) is required for simulation of the schematic's HDL functional model.

Note Detailed information on using ECS can be found in the ECS online help.

The ECS Window

The ECS window is the main interface for the ECS tools. To open the ECS window from Project Navigator:

- Create a new schematic source for the project (see the "Creating a Schematic Source File" section below).

  OR

- Double-click an existing schematic source file (.sch) in the Sources in Project window.

You edit both schematics and symbols in the ECS window. You edit a schematic in a schematic window and a symbol in a symbol window within the ECS window. An ECS window containing both a schematic and a symbol is shown in the following figure.

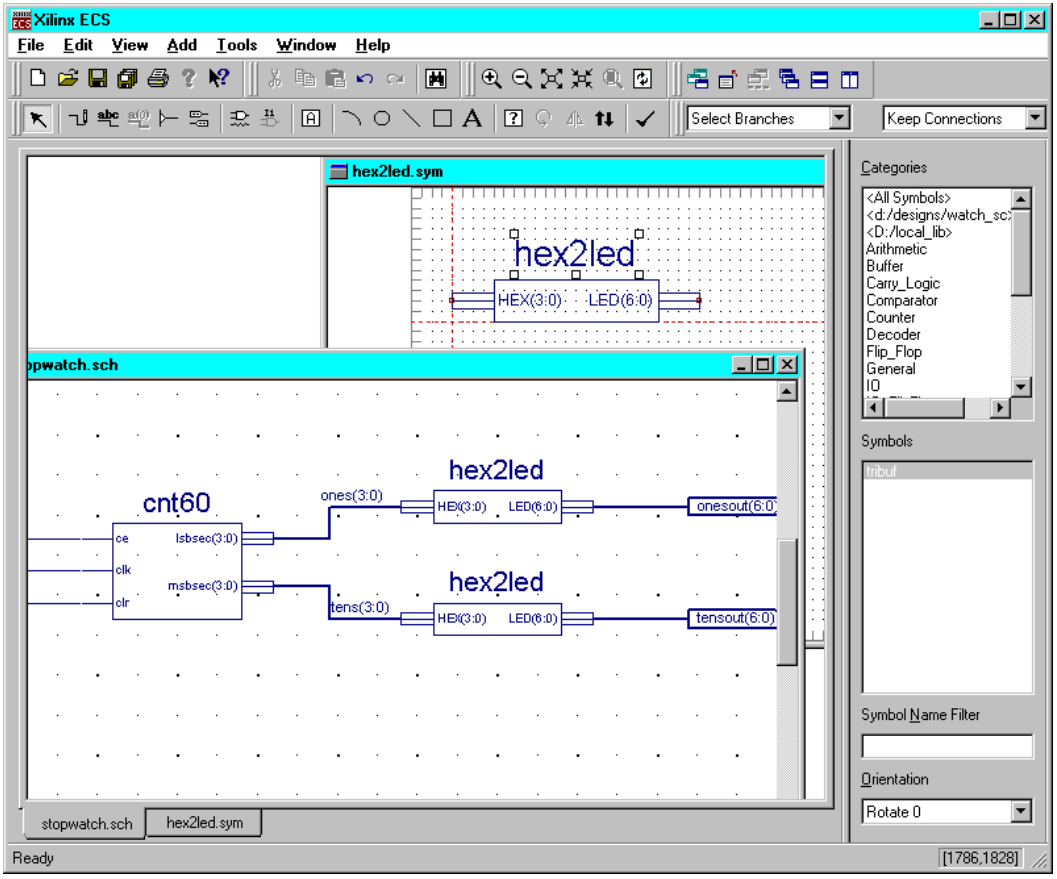

Figure 7-4 ECS Window with Schematic and Symbol

The ECS main window contains the following:

- Title Bar

  The title bar displays the name of the application and the name of the schematic or symbol you are editing.

- Menu Bar

  The menu bar contains the drop-down menus that control the operation of ECS. The menu items in the drop-down menus perform the ECS commands. ECS has seven menus: File, Edit, View, Add, Tools, Window, and Help.

Some menu items change depending on whether you are editing a schematic or a symbol. Most menu commands can be performed on either a schematic or a symbol.

- Toolbars

  This area contains six separate toolbars:

  - File
  - Edit
  - View
  - Window
  - Tools
  - Options

  The File, Edit, View, Window, and Tools toolbars contain buttons that provide access to frequently used commands. The options toolbar, which changes depending on the command you are performing, contains options for the current command.

- Workspace

  The workspace contains the schematic and symbol drawings that you edit using ECS. Each drawing appears in its own window in the workspace.

  You can have multiple schematic windows and multiple symbol windows in the workspace. You can also have schematic windows and symbol windows in the workspace at the same time. Tabs at the bottom of the workspace show all of the windows in the workspace and allow you to select the window to view.

- Schematic Windows

  Schematic windows contain the schematic drawings you edit in ECS.

  Tabs at the left side of each schematic window show all of the pages in the schematic and allow you to select the page to view.

- Symbol Windows

  Symbol windows contains the symbol drawings you edit in ECS.

- Symbol Browser

  The symbol browser allows you to select symbols to instantiate into a schematic. You can select symbols from the Xilinx symbol libraries supplied with ECS, or from symbol libraries you create.

  The symbols are divided into categories (for example, decoders and shift registers) to make it easier to find the symbol you want.

- Status bar

  The status bar displays command and processing information.

Concepts Required to Use ECS

A schematic created in ECS is composed of the following items:

- Symbols
  These can be symbols from the standard Symbol libraries, symbols representing other schematics you have drawn (Block symbols), or symbols you have created from scratch.
- Wires
  Wires connect the symbols. They can be single-signal (*nets*) or multiple-signal (*buses*).
- I/O Markers
  I/O markers show where signals enter or exit the schematic, and the direction (*polarity*) of the signal (that is, whether it's an input, output, or bidirectional).
- Graphics & Text
  Graphics and text are usually added to display explanatory data. They are optional and have no electrical meaning.

A valid schematic must contain at least the first three components: symbols, wires, and I/O markers. For instance, a single, isolated component symbol cannot be the only element in a schematic. The schematic must include I/O markers for the external connections to the schematic, and these markers must be connected to the symbol with wires.

Note HDL keywords cannot be used for names of items on a schematic.

Symbols

Symbols are graphic representations of components. The term *symbol* usually refers to an electrical symbol, such as a gate or a subcircuit. You can also create graphic-only symbols (such as title blocks) in ECS, but these have no electrical meaning.

Each schematic symbol is stored in a file ending with a **`.sym`** extension, or may be included in a library file with a **`.lib`** extension. The symbol contains four types of information:

- "Graphics"
- "Text"
- "Pins"
- "Attributes"

Graphics

Graphics are pictures of the symbols.They tell ECS how to draw the symbol. Symbol graphics have no electrical meaning, showing only the position of the component in the circuit. The electrical behavior of a symbol is defined by its attributes and pins, not the graphics that represent it. Explanatory or descriptive text displayed with a symbol is also considered *graphic* information without electrical meaning.

Text

Text labels the symbol, or adds supplemental information.

Pins

Pins provide electrical connection between the symbol and the schematic's wiring. Symbol pins are the connecting points between the symbol and the schematic wiring. If the symbol represents an individual component, the symbol pin represents the physical pin where a conductor can be attached. If the symbol is a block symbol, the symbol pin represents a connection to an internal net of the design unit represented by the block symbol.

Pins can either represent a single electrical connect point (a scalar pin) or multiple electrical connect points (a bus pin).

Note You can add only one kind of pin (using **Add** → **`Pin`** in a symbol window). Whether it is a scalar pin or a bus pin depends on the name

you give the pin using the pin name attribute. Bus pins are named as *busname*[*numberlist*] where *busname* is the name of the bus and *numberlist* is a list of numbers separated by:

- commas
  example: [1,3,5]
- a range of numbers separated by a colon
  example: [8:15]
- both
  example: [1,3,5,8:15]

Attributes

Attributes describe the symbol's electrical behavior, the symbol's component parts (for example, its pins), and other characteristics. Each symbol has a number of predefined attributes that describe its symbol name, symbol type, and other unchanging characteristics. Other attributes can be given values after the symbol is placed in the schematic. These attributes can have different values for each symbol instance, which permits detailed customization of a design.

A symbol's attribute set is the most important part of the Symbol. Without attributes, simulation and modeling programs would know nothing about the electrical behavior of the symbol.

Attributes associate data with symbols, pins, and nets. (*Nets* are schematic wiring.) The data describe the electrical characteristics (or other properties) of the symbols and their pins.

An attribute has a name and a value. You can assign or change the values of most attributes at any point in the development process. You can assign some attributes fixed values that cannot change. You can assign, change, or override other attributes later in development. If an attribute value is assigned to a symbol, it becomes the default value and is used with every instance of that symbol.

ECS attributes can also be set on schematic components for implementation processing. See the "Xilinx Implementation Attributes and Constraints" section for information.

Wires (Nets and Buses)

Wires are the lines that electrically connect the symbol pins. Symbol pins are the only connection points for wires. You cannot connect wires to the symbol body itself.

Wire Types

There are two types of wires:

- Scalar, which represent a single electrical connection
- Buses, which represent multiple electrical connections

Click **Add** → **Wire** to add wires to schematics.

Bus Taps

Bus taps can be added using the **Add** → **Bus Tap** command. A bus tap allows you to extract an individual net of a bus and connect it to a scalar pin on a symbol. Alternatively, you can connect multiple nets from a bus or an entire bus to a bus pin on a symbol, assuming that the size of the tapped bus and the size of the bus pin (that is, the number of nets that they contain) is the same.

Buses are most often used to group related signals. However, a bus can be any combination of signals, related or not. Buses are especially useful for routing a large number of signals from one side of the schematic to the other.

Buses also make it possible for a single I/O marker to connect more than one signal to a Block symbol. The signal names don't have to match, but both pins must carry the same number of signals.

You can add only one kind of wire (using **Add** → **Wire**). Whether it is a net or a bus depends on how you name the wire (using the **Add** → **Net Name**).

Buses are named as *busname*(*numberlist*) where:

- *busname* is the name of the bus
- *numberlist* is a list of numbers separated by:
  - commas
    example: (1,3,5)
  - a range of numbers separated by a colon
    example: (8:15)

- both commas and a colon
  example: (1,3,5,8:15)

Wires and Net Names

Wires are used to connect symbol pins on a schematic. Every wire has a net name, which serves to identify the wire to ECS and netlister programs. ECS automatically supplies a name when you add the first wire of a net to the schematic. This default name is of the form XLXN_*n* (where *n* is an integer), and you can change this name later.

Two or more wires may have the same net name. Each wire that shares a common net name becomes part of a single net, and all symbol pins connected to these wires are electrically connected.

To illustrate this concept, consider the following schematic fragment.

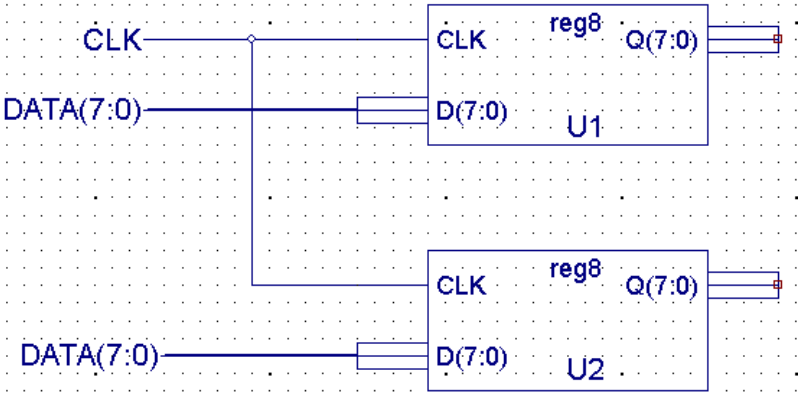

Figure 7-5 Schematic Fragment

In this example, a single wire, with the net name of CLK, is connecting the pins CLK of symbol instances U1 and U2. And two wires, with the common name of DATA[7:0], are being used to connect the bus pins named D[7:0] on U1 and U2 (this is also an example of buses and bus pins).

You would normally rename all wires that connect to inputs or outputs and any *internal* nets with signals you want to view during simulation. You can use any name you like, but you usually choose a name that suggests the name or function of the signal carried by that wire.

Net Attributes

Like symbols and symbol pins, nets can also have attributes. Net attributes can describe characteristics associated with nets. Good examples are the KEEP and SAVE attributes.

Note Nets are the wiring that connects symbols to each other and makes external connections.

I/O Markers

I/O markers are used to flag the nets that are inputs, outputs, or bidirectional signals in the schematic. If a net name appears multiple times on a schematic, only one instance of the net name needs to be flagged with an I/O marker.

I/O markers are added to schematics using the **Add** → **I/O** Marker command. When you execute **Add** → **IO Marker**, the options toolbar displays these choices.

Figure 7-6 Options Menu for Add I/O Markers Command

Click the radio button corresponding to the type of I/O marker you wish to add. You can then add I/O markers by:

- Clicking the ends of individual nets that you wish to flag
- Dragging a box around the net ends

I/O Markers and Block Symbols

If the schematic has a corresponding block symbol, each net flagged with an I/O marker should also have a corresponding pin on the block symbol for the schematic.

The following conditions would cause errors when you try to create an HDL model for the schematic:

- The block symbol has a pin with a given name, but the schematic for that symbol does not have a net with that same name.
- The schematic has a net with the appropriate name, but the net is not flagged with an I/O marker.

Graphics

Although symbols, wires, and I/O markers are visible, graphical items, they also have a functional or electrical meaning. In the context of this section, *graphics* refers to the non-functional graphical parts of the schematic.

For example, you might add graphics showing the expected waveforms at different points in the circuit. Alternatively, you could draw the company's logo and add it to each schematic for identification.

The most common use of graphics is to create a title block. The block shows the name and address of your company, and can include your company logo and blank spaces for the project name, schematic sheet number, and other items.

The title block is a symbol. A title block template (the *tblock* symbol) is located in the General symbol library. See the online Help for detailed information on title blocks.

Text

Text, like graphics, can provide additional information about the schematic or its project. Text can be placed anywhere on a schematic, even if it overlaps symbols or wires. Use **Add** → **Text** to add text to schematics and symbols.

ECS Menu Commands

The following list describes each ECS menu. To view detailed help information for each menu item:

1. Click a menu.
2. Move the cursor to highlight a menu item in the menu.
3. Press F1.

File Menu

Use the File menu to:

- Create, open, and save schematics and symbols
- Print a schematic or symbol drawing
- Exit ECS

Edit Menu

Use the Edit menu to:

- Undo and Redo commands
- Cut, copy, paste, and delete objects
- Open a symbol window to edit a symbol
- Update obsolete symbols
- Rename buses
- Select objects
- Find nets and instances
- Define, edit, and view attributes for pins, symbols and nets
- Set ECS preferences

View Menu

Use the View menu to:

- Control the display of toolbars and screen areas
- Zoom and pan the display
- Push and Pop

Add Menu

Use the Add menu to:

- Add wires, net names, bus taps, and I/O markers
- Add and modify symbols and symbol instances
- Display attributes on instances and nets
- Add graphics (arcs, circles, lines, rectangles, and text) to a schematic

Tools Menu

Use the Tools menu to:

- Check a schematic or symbol for errors
- Automatically create a symbol representing a schematic
- Automatically generate a symbol drawing in a symbol window
- Automatically generate I/O markers in a new schematic
- Get information about objects by running a query

Window Menu

Use the Window menu to:

- Open and close editing windows
- Display windows in different ways

Help Menu

Use the **Help** menu to:

- View the help contents
- Open the online document collection
- View ECS revision information

Editing Schematics in ECS

The following sections demonstrate some basic ECS schematic entry tasks.

Adding a Symbol

To add an existing symbol to your schematic:

1. Click **Add** → **Symbol**.
2. In the symbol browser, click a library or symbol category in the Categories box to display its symbol list in the Symbols box.

   The local library contains the symbols you created in ECS for the project. The Xilinx supplied libraries are divided into component categories. Click **All Symbols** to view an alphabetical list of all available symbols; this includes the Xilinx-supplied libraries and the local library. See the "Using Symbols from Other Projects" section for information on making more symbols available in the symbol browser.
3. Click the symbol in the symbol browser and drag it to the schematic for placement.

   Note To make it easier to locate the symbol you are looking for, type the first few letters of the symbol name in the Symbol Name Filter box. The Symbols box is updated to show only the symbols that begin with those letters, and you can select the symbol you want from this shorter list.
4. Click to place the symbol.

   You can continue to select and place symbols or press **Esc** to exit the Add Symbol mode.

As you place each symbol, ECS automatically gives the symbol a unique *instance name* of the form XLXI_*nn* (where *nn* is an integer). The instance name identifies the symbol to ECS and netlister programs. To change the instance name, use **Add** → **Instance Name**. ECS does not allow you to repeat an existing name.

ECS lets you define an iterated instance in which a single symbol represents many instances of that symbol. An iterated instance is created by giving a symbol instance an instance name that includes a

[*numberlist*]. For example, the instance name IA[7:0] creates eight instances of the corresponding symbol.

Adding a Wire

To add a wire:

1. Click **Add** → **Wire**.
2. In the options toolbar, select **Automatic**.
3. Click the wire's start point.

   A wire can start and end at a symbol pin, another wire, or an empty point in the schematic. For some points (for example, symbol pins and wire corners) four red squares appear to help you locate the exact point.
4. Move the cursor to the wire's end point in the schematic and click the desired end point.

   A wire segment is automatically routed between the two points. If the second point was a *red square* point, the Add Wire command ends.
5. If the second point was not a *red square* point, continue clicking points to draw additional connected wire segments.
6. Double-click the last point to connect.

You can continue to add wires, or press **Esc** to exit the Add Wire mode.

Moving a Wire

You can move a wire with or without disconnecting it, depending on whether you select **Keep Connections** or **Break Connections** in the options toolbar.

Moving a Wire Without Disconnecting

To move a wire without disconnecting it:

1. Click **Edit** → **Select**.
2. In the options toolbar, click **Select Wires** in the first box and **Keep Connections** in the second box.

3. Move the cursor to the wire.
4. Click and hold the left mouse button.
5. Drag the wire to its new location.

Moving and Disconnecting a Wire

To disconnect a wire when you move it:

1. Click **Edit** → **Select**.
2. In the options toolbar, click **Select Wires** in the first box and **Break Connections** in the second box.
3. Move the cursor to the wire.
4. Click and hold the left mouse button.
5. Drag the wire to its new location.

Removing a Symbol or Other Object

To remove a symbol or other object from a schematic:

1. Select the symbol or object.
2. Click **Edit** → **Delete**.

Panning

Use the following commands or keyboard shortcuts to pan a schematic or symbol window.

Table 7-1 Panning

Pan ...	Menu Command	Arrow key
Left	**View** → **Pan** → **Left**	left (←)
Right	**View** → **Pan** → **Right**	right (→)
Up	**View** → **Pan** → **Up**	up (↑)
Down	**View** → **Pan** → **Down**	down (↓)

Zooming

Use the following commands or keyboard shortcuts to zoom the display in a schematic or symbol window.

Table 7-2 Zooming

Zoom ...	Menu Command	Function Key
In	**View** → **Zoom** → **In**	**F8**
Out	**View** → **Zoom** → **Out**	**F7**
Full View (display the entire schematic or symbol drawing in the window)	**View** → **Zoom** → **Full View**	**F6**
Box (fill the window with an area that you select by drawing a box around it)	**View** → **Zoom** → **To Box**	None
Selected (center the selected objects in the window and display them at the highest magnification)	**View** → **Zoom** → **To Selected**	None

Editing Symbols in ECS

Use ECS to create and edit symbols and symbol attributes. Many of the operations performed in a symbol window are initiated in a schematic window.

Opening a Symbol Window

To open a symbol window:

1. In a schematic window, select an instance of the symbol you want to modify.
2. Click **Symbol** → **Edit**.

   A symbol window opens with the selected symbol.

   Note If you selected a symbol from the read-only Xilinx-supplied libraries, a dialog box opens before the symbol window opens to

allow you to confirm copying the symbol to the Local library for modification.

To open a symbol window to create a new block symbol, click **File** → **New** and specify that you want to create a new symbol.

Symbol Types

ECS creates three different kinds of symbols:

- "Block Symbols"
- "Graphic Symbols"
- "Master Symbols"

Block Symbols

Block symbols are the basic symbols you use to build your design. The symbols in the Xilinx-provided libraries are mainly Block symbols. A Block symbol represents both primitives and macros (schematics at the next-lower level of the hierarchy). Pins are permitted only on Block symbols.

All Block symbols have the same basic design: a rectangle with pin leads extending outward. The rectangle's height and width are automatically scaled according to the number of pins and the length of their names. The input pins are placed on the left side and the output pins are placed on the right side.

A Block symbol has an attribute window near the top for displaying the name, and a window near the bottom for displaying the instance name.

Graphic Symbols

Graphic symbols add information that is not part of the circuitry. Graphic symbols are typically used for tables and notes. Pins are not associated with Graphic symbols. Graphic symbols are never included in the design hierarchy or netlists.

Master Symbols

Master symbols are used for title blocks, logos, revision blocks, and other standardized graphic symbols. You can add text to a Master

symbol to display the company name, address, and project description, date, and other items.

Master symbols do not have pins. They are never included in the design hierarchy or netlists.

Symbol Libraries

ECS automatically includes the symbol libraries specific to the targeted Xilinx device. Symbols that you create are added to the project's Local symbol library. You can also access the Local symbol libraries from other projects to use those symbols in your project.

All libraries and symbols available for schematic entry are listed in the symbol browser opened from **Add** → **Symbol**.

Modifying an Existing Symbol

You can modify symbols in the Symbol libraries only if they are in the Local directory.

To modify a symbol in the Local directory:

1. In a schematic window, select an instance of the symbol you want to modify.
2. Click **Symbol** → **Edit**.

   A symbol window opens with the selected symbol.

   Note If you selected a symbol from the read-only Xilinx-supplied libraries, a dialog box opens before the symbol window opens to allow you to confirm copying the symbol to the Local library for modification.
3. Edit the symbol.
4. Save the symbol.
5. Exit the symbol window.

When you return to the schematic window containing the symbol you edited, a dialog box appears to allow you to update the symbol in the schematic window.

Creating a New Block Symbol

ECS automatically creates a new Block symbol in an empty symbol window.

To create a new Block symbol:

1. Open a new symbol window.
2. Click **Tools** → **Create Symbol**.
3. In the Create Symbol dialog box, enter pin names (separated by commas) in the Inputs, Outputs, and Bidirection fields. You can use bus notation (D[7:0], for example) when entering a pin name, as shown in the following figure.

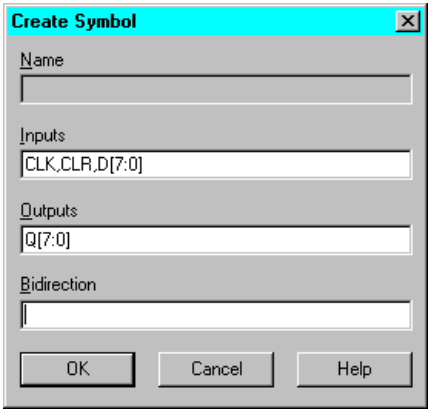

Figure 7-7 Create Symbol Dialog Box

4. Click **OK** to create the new symbol.

   ECS creates the symbol, as shown in the following figure. Bus pins are drawn with rectangles around the wires between the symbol body and the pins.

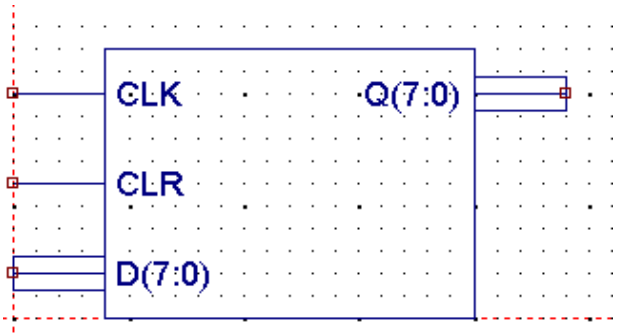

Figure 7-8 New Block Symbol

5. When you save the new symbol, you will have to specify the file name for the symbol. The file name becomes the name of the symbol.

Creating a Block Symbol from a Schematic

To create a new block symbol representing a schematic in the ECS window:

1. Click **Tools** → **Create Symbol**.

   A Create Symbol dialog box opens. The schematic name is automatically placed in the Name field. The pins created in the schematic are automatically entered in the appropriate pin fields (**Inputs**, **Outputs**, and **Bidirection**).

2. Click **OK** to create the new symbol and add it to the Local library.

Creating a Symbol from an HDL Source

To create a schematic symbol for an HDL source:

1. In Project Navigator, click the HDL module in the Sources in Project window.
2. Double-click the **Create Schematic Symbol** process in the Processes for Current Source window.
3. The symbol file (.sym) is automatically created and added to the Local symbol library for placement on a schematic.

Using Symbols from Other Projects

You can add a search path to another project to have the local symbols in that project added to the symbol browser. You include the project's symbols in the symbol browser on a device basis. The symbols are available only for projects that target a specified device. Only symbols created in ECS are available for use in other projects.

To add the local symbols created in an ISE 4.x project to the symbol browser for all projects that target a specified device family:

1. In ECS, click **Edit** → **Preferences**.
2. In the Device Families page of the Preferences dialog box, click the plus sign next to the name of the device family for which you will add symbols.
3. For the selected device family, click **Symbol Library Files and Paths**.
4. Click the **Edit** button.

   A dialog box appears for you to locate the desired directories and files.
5. In the **File or Path to Add** field, enter the path to the project whose local symbols are to be added to the symbol browser.

   OR

   Click the **Browse Directory** or **Browse File** button and use the dialog box that appears to browse to the project containing the symbols you want to make available to other projects targeting the same device. Then click the OK button to return to the previous dialog box.

   The full path to the selected project appears in the **Path** field of the dialog box.
6. Click the **Add** and **OK** buttons to add the local symbols for the selected project (specified in the **File or Path to Add** field) to the search path for all projects that use the specified device (for example, Virtex).

The symbols in the selected project are now available in the symbol browser for all projects that use the selected device.

Note When you use symbols from other projects, be sure to edit the symbol and use the **Save As** command to place it in the Local library of the current project.

Guidelines for Creating Schematics

All schematics are netlisted to VHDL or Verilog structural netlists, then processed by either the XST or FPGA Express synthesis engine. Because of this, you must follow these guidelines when creating schematic sources:

- Do not use HDL keywords for net or instance names.
- Use I/O primitives only.

  Use I/O primitives such as IBUF, OBUF, IFD, IFDX, OFD, OFDX only. Do not use I/O macros such as IBUF4, IFD_1. If I/O macros are used, the synthesis engine inserts an additional IBUF or OBUF at the port causing errors in the MAP process.
- (For VHDL Only) All input pins of the Xilinx unified library components must be connected.

  Failure to connect all inputs pins of library components results in errors during synthesis due to the discrepancy between the library component declaration and the actual use.
- (For FPGA Express Only) Do not use Carry Logic primitives on schematics in the FPGA Express flow.

  Carry Logic is re-optimized by FPGA Express. This may result in different logic than intended for the design.
- (For FPGA Express Only) Specify global buffers pins via Express Constraints Editor rather than via the schematic.

  Use a generic IBUF on clock pins. Then use the Express Constraints Editor to select a global buffer for the desired ports. Failure to use this method causes errors in MAP due to the clock pin being left unconnected.

  To open the Express Constraints Editor, click Edit Constraints under Synthesis in the Processes for Current Source window.

Chapter 8

LogiBLOX

This chapter contains the following sections:

About LogiBLOX

LogiBLOX is an onscreen design tool for creating high-level modules such as counters, shift registers, and multiplexers for XC4000, Spartan, and SpartanXL FPGA designs and 9500/XL/XV CPLD designs.

LogiBLOX includes both a library of generic modules and a set of tools for customizing these modules. LogiBLOX modules are pre-optimized to take advantage of Xilinx architectural features such as Fast Carry Logic for arithmetic functions and on-chip RAM for dual-port and synchronous RAM. With LogiBLOX, you can create high-level LogiBLOX modules that fit into your schematic-based design or HDL-based design.

Note LogiBLOX is not available for use with Virtex, VirtexE, Virtex2, Spartan2, and Spartan2E devices. The CORE Generator supports those devices. See the "CORE Generator" section of the "Design Flow" chapter.

Starting LogiBLOX

You can start LogiBLOX as a standalone program or as an integrated design entry tool in Project Navigator.

To start LogiBLOX as a standalone program, click **Start** → **Programs** → **ISE 4.x** → **Accessories** → **LogiBLOX** from your PC's desktop.

Note Your startup path is set during installation, and may differ from the path shown above.

Within an ISE 4.x project, start LogiBLOX to create a core for use in a design as follows:

1. Open Project Navigator.
2. Click **Project** → **New Source**.
3. In the New dialog box, click **LogiBLOX Module**.
4. Enter a file name for the new module.
5. Click **Next**.
6. In the New Source Information dialog box, click **Finish**.
7. LogiBLOX opens. See the following figure.

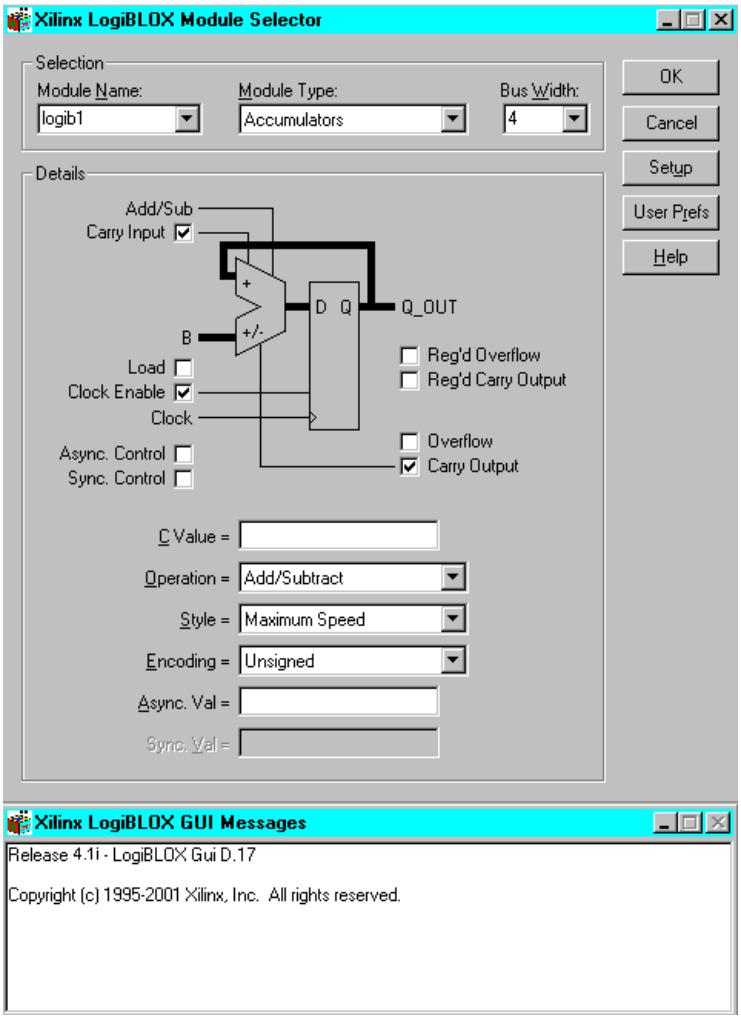

Figure 8-1 LogiBLOX Module Selector - Accumulators

LogiBLOX Setup

When you open LogiBLOX from an ISE 4.x project, the project information is automatically entered in the LogiBLOX Setup menu. The Vendor Name, Project Directory, and Device Family information are all entered based on that project. To open the LogiBLOX Setup dialog box, click **Setup** on the LogiBLOX Module Selector dialog box.

You can instantiate a LogiBLOX module in VHDL or Verilog code. By default, the language designated in the project's synthesis tool determines which template (VHDL or Verilog) LogiBLOX creates for the project. To change this, use the Options tab to select the type of simulation netlist and component declaration template (VHDL or Verilog) you need. For VHDL, click **VHDL template** and **Behavioral VHDL netlist.** See the following figure. For Verilog, click **Verilog template** and **Structural Verilog netlist**.

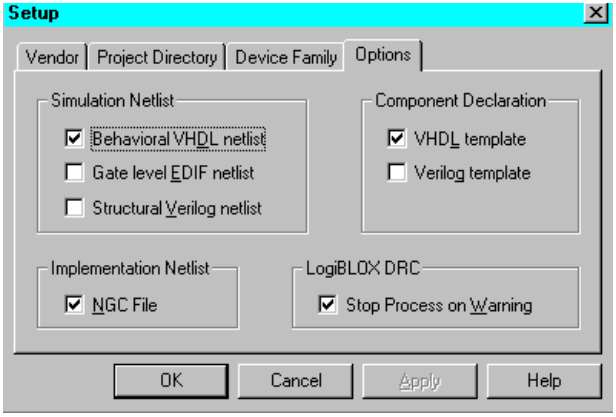

Figure 8-2 LogiBLOX Setup Window

Creating LogiBLOX Modules

After you have opened LogiBLOX from your ISE 4.x project, create a new module as follows:

1. The name you entered in the Project Navigator's New source appears in the Module Name field. You need not enter a name.

   The Module Name shown here is used as the name of the instantiation in the HDL code.

2. Select the type of module from the Module Type list box. The initial LogiBLOX window shows the fields necessary for creating an accumulator module. See the following figure. Click the down arrow in the Module type field to display the list of module types you can create.

   Following is a list of the LogiBLOX modules:

   - Accumulator
   - Adder/Subtracter
   - Clock Divider
   - Comparator
   - Constant
   - Counter
   - Data Register
   - Decoder
   - Input/Output (schematic only)
   - Memory
   - Multiplexer
   - Pad (schematic only)
   - Shift Register
   - Simple Gates
   - 3-state Buffers

3. Select a module type and define its attributes. The following figure shows the LogiBLOX dialog box for Module Type = Memories.

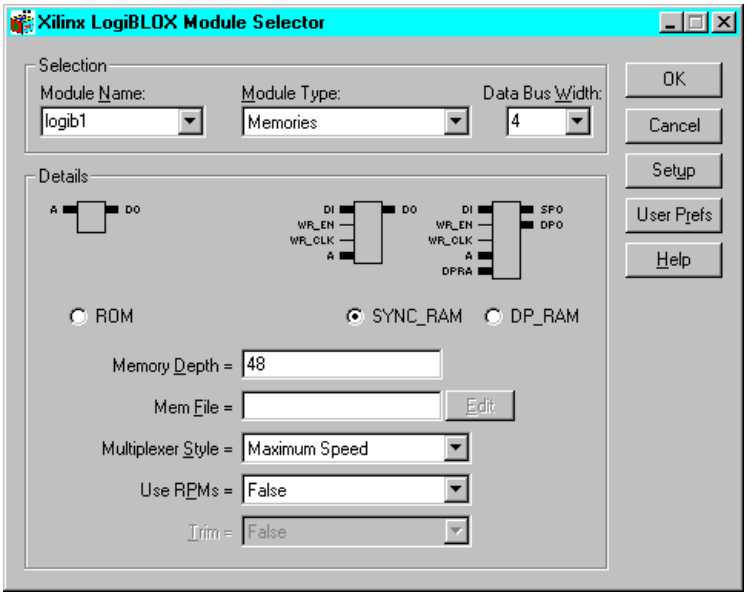

Figure 8-3 LogiBLOX Dialog Box for Module Type = Memories

4. Select the bus width for the module from the Bus width list box.
5. Select or deselect optional pins of the module symbol displayed in the Details box by clicking the appropriate check boxes.
6. Click **OK**. The LogiBLOX module and its related files are created, and the LogiBLOX .mod file is automatically added to the ISE 4.x project as a user document.

The LogiBLOX module is a collection of several files including those listed below. The files are located in your ISE 4.x project directory.

Note Whether the VHDL files (.vhi, .vhd) or Verilog files (.vei, .v) are created depends on the setting in the LogiBLOX Setup Options tab.

component_name.ngc	Netlist used during the Translate phase of Implementation
component_name.vei	Instantiation template used to add a LogiBLOX module into your Verilog source code
component_name.v	Verilog file used for functional simulation

component_name.vhi	Instantiation template used to add a LogiBLOX module into your VHDL source code
component_name.vhd	VHDL file used for functional simulation
component_name.mod	Configuration information for the module
logiblox.ini	LogiBLOX configuration for the project

The component name is the name given to the LogiBLOX module in the LogiBLOX Module Selector GUI. The port names are the names provided in the instantiation template file.

Using LogiBLOX Modules in ISE 4.x Projects

If you initiated LogiBLOX from the Project Navigator New window, the LogiBLOX module and its related files are placed in the current ISE 4.x project directory. The related files include schematic, VHDL, and Verilog templates that can be used in your project. The LogiBLOX module file (.mod) is added as a user document and appears in the Sources in Project window.

Editing LogiBLOX Modules

To edit a LogiBLOX module:

1. Double-click the module name in the Sources in Project window.
2. The LogiBLOX Module Selection window opens with the selected module loaded.

Using LogiBLOX Modules in Schematic Sources

You can use LogiBLOX components in schematics as well as HDL designs for FPGAs and CPLDs. Once you are in the LogiBLOX GUI, you can customize standard modules and process them for insertion into your design.

To use LogiBLOX in schematic sources:

1. Add the *logiblox_module*.v (Verilog) or .vhd (VHDL) file using **`Project`** → **`Add Source`**.

2. Click the newly added *logiblox_module*.v or .vhd file in the Sources in Project window.
3. Click **`Create Schematic Symbol`** in the Processes for Current Source window.

   A symbol (*logiblox_module*.sym) is created for the *logiblox_module*.v or .vhd module.

   The *logiblox_module* symbol is added to the local Symbol Library of the Engineering Capture System (ECS) for use in schematic sources.

   For synthesis, FPGA Express treats the LogiBLOX modules in schematic sources as black boxes because of the `synopsys translate off` directive in the .v and .vhd files generated by LogiBLOX.

Note Once a LogiBLOX module is created, do not change parameters for the module on the schematic. Any changes to the module parameters must be made through the LogiBLOX GUI and a new module created.

Instantiating LogiBLOX Modules in an HDL Source

You can instantiate the LogiBLOX components in your HDL code to take advantage of their high-level functionality. Define each LogiBLOX module in HDL code with a component declaration, which describes the module type, and a component instantiation, which describes how the module is connected to the other design elements.

LogiBLOX modules may be generated in ISE and then instantiated in the VHDL or Verilog code. This flow may be used for any LogiBLOX component, but it is especially useful for memory components such as RAM. Never describe RAM behaviorally in the HDL code, because combinatorial feedback paths will be inferred.

VHDL Instantiation

When you instruct LogiBLOX to create a VHDL component in an ISE 4.x project, a VHDL instantiation template file is automatically placed in the project directory when the LogiBLOX module is saved. The template (.vhi file) is then available to use to instantiate the LogiBLOX module into a VHDL source for the project.

To instantiate a LogiBLOX module into a VHDL source in an ISE 4.x project:

1. Open Project Navigator.

   Note The example used in this section is for a LogiBLOX RAM48X4S memory component named *logib1*.

2. Click **File** → **Open**
3. In the Open dialog box, select the VHDL instantiation template (.vhi file) for the LogiBLOX module from the project directory.
4. The VHDL instantiation template opens in the HDL Editor workspace.

The following figure shows an example for a LogiBLOX memory module named *logib1*.

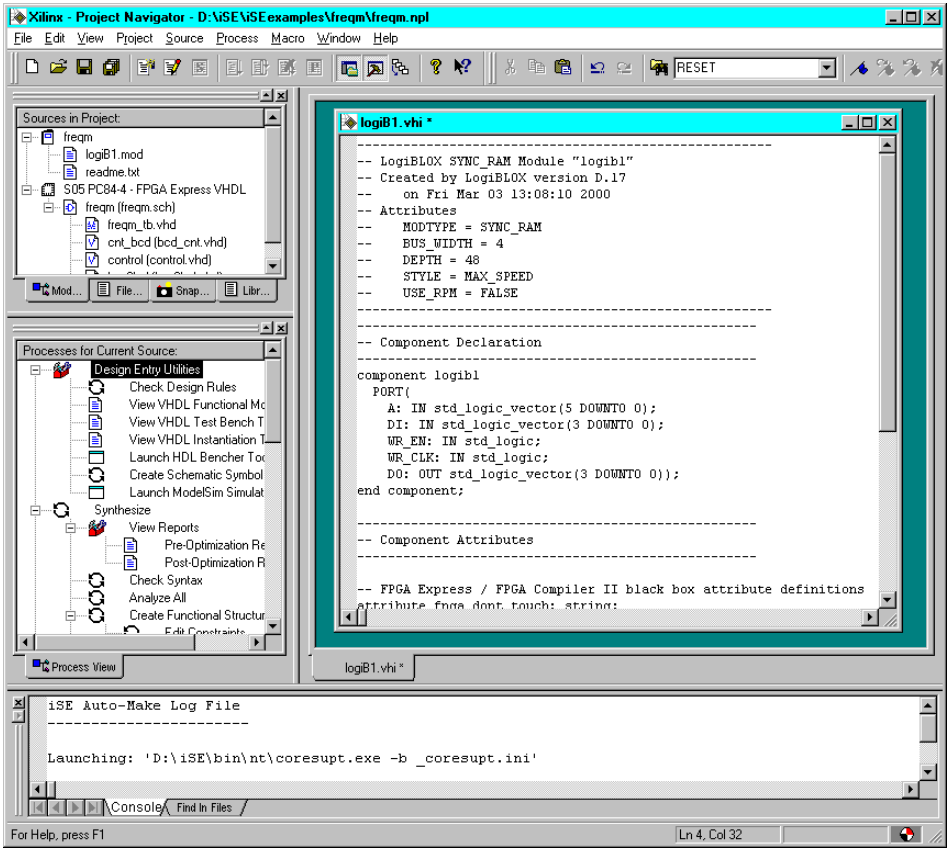

Figure 8-4 Example for a LogiBLOX Memory Module Named "logib1"

The complete text of the LogiBLOX VHDL instantiation template for logib1.vhi is as follows.

```
------------------------------------------------
-- LogiBLOX SYNC_RAM Module "logib1"
-- Created by LogiBLOX version D.17
--    on Fri. Mar 03 13:08:10 2000
-- Attributes
--    MODTYPE = SYNC_RAM
```

```
--    BUS_WIDTH = 4
--    DEPTH = 48
--    STYLE = MAX_SPEED
--    USE_RPM = FALSE
-----------------------------------------------
-----------------------------------------------
-- Component Declaration
-----------------------------------------------
component logib1
  PORT(
    A: IN std_logic_vector(5 DOWNTO 0);
    DI: IN std_logic_vector(3 DOWNTO 0);
    WR_EN: IN std_logic;
    WR_CLK: IN std_logic;
    DO: OUT std_logic_vector(3 DOWNTO 0));
end component;

-----------------------------------------------
-- Component Attributes
-----------------------------------------------

-- FPGA Express / FPGA Compiler II black box
-- attribute definitions
attribute fpga_dont_touch: string;
attribute fpga_dont_touch of logib1:component is
   “true”;
-- XST black box attribute definitions
attribute box_type: string;
```

```
attribute box_type of logib1:component is
   “black_box”;


-----------------------------------------------

-- Component Instantiation

-----------------------------------------------

instance_name : logib1 port map

(A => ,

 DI => ,

 WR_EN => ,

 WR_CLK => ,

 DO => );
```

5. Click **File** → **Open** on the Project Navigator menu to open the VHDL file in which the LogiBLOX component is to be instantiated.
6. Cut and paste the Component Declaration from the LogiBLOX component's .vhi file to your project's VHDL source, placing it after the architecture statement in the VHDL code.
7. Cut and paste the Component Instantiation from the LogiBLOX component's .vhi file to your VHDL source code after the "begin" line. Give the inserted code an instance name. Edit the code to connect the signals in the design to the ports of the LogiBLOX module.

   Following is an example of VHDL source code with the LogiBLOX instantiation for the component named *logib1*. For each NGC file from LogiBLOX, you may have one or more VHDL files with the NGC file instantiated. In this example, there is only one black box instantiation of memory, but multiple calls to the same module may be done.

```
library IEEE;

use IEEE.std_logic_1164.all;

use IEEE.std_logic_arith.all;
```

```
entity top is
port (    D: in STD_LOGIC; CE: in STD_LOGIC;
        CLK: in STD_LOGIC; Q: out STD_LOGIC;
       Atop: in STD_LOGIC_VECTOR (5 downto 0);
      DOtop: out STD_LOGIC_VECTOR (3 downto 0);
      DItop: in STD_LOGIC_VECTOR (3 downto 0);
   WR_ENtop: in STD_LOGIC;
  WR_CLKtop: in STD_LOGIC);
end top;

architecture inside of top is

component userff
port (     D: in STD_LOGIC; CE: in STD_LOGIC;
         CLK: in STD_LOGIC; Q:  out STD_LOGIC);
end component;

component memory
port ( A:  in STD_LOGIC_VECTOR (5 downto 0);
      DI: in STD_LOGIC_VECTOR (3 downto 0);
   WR_EN:  in STD_LOGIC;
  WR_CLK:   in STD_LOGIC;
      DO:  out STD_LOGIC_VECTOR (3 downto 0));
end component;

begin
```

```
UO:userff port map (D=>D, CE=>CE, CLK=>CLK,
   Q=>Q);


U1:memory port
   map(A=>Atop,DI=>DItop,WR_EN=>WR_ENtop,

        WR_CLK=>WR_CLKtop, DO=>DOtop);

end inside;
```

8. Save the VHDL source file. The design with the instantiated LogiBLOX module can now be checked for syntax errors and synthesized.

   Note When the design is synthesized, a warning is generated that the LogiBLOX module is unlinked. Modules instantiated as black boxes are not elaborated and optimized. The warning message is just reflecting the black box instantiation.

Verilog Instantiation

When you instruct LogiBLOX to create a Verilog component in an ISE 4.x project, a Verilog instantiation template file is automatically placed in the project directory when the LogiBLOX module is saved. The template (.vei file) is then available to use to instantiate the LogiBLOX module into a Verilog source for the project.

Use the following procedure to instantiate a LogiBLOX module into a Verilog source in an ISE 4.x project. The example used in this section is for a LogiBLOX RAM48X4S memory component named *logib1*.

1. Click **File** → **Open** from the Project Navigator menu.
2. In the Open dialog box, select the Verilog instantiation template (.vei file) for the LogiBLOX module from the project directory.

   The VHDL instantiation template opens in the HDL Editor workspace.

   The complete text of the LogiBLOX Verilog instantiation template for logib1.vei is as follows.

```
//-----------------------------------------------

// LogiBLOX SYNC_RAM Module “logib1”
```

```
// Created by LogiBLOX version D.17
//    on Fri. Mar 03 14:26:15 2000
// Attributes
//    MODTYPE = SYNC_RAM
//    BUS_WIDTH = 4
//    DEPTH = 48
//    STYLE = MAX_SPEED
//    USE_RPM = FALSE
//-----------------------------------------
logib1  instance_name
( .A(),
  .DO(),
  .DI(),
  .WR_EN(),
  .WR_CLK());

// FPGA Express / FPGA Compiler II black box
// attribute definitions
// synopsys attribute fpga_dont_touch “true”
// synthesis attribute fpga_dont_touch of
   instance_name is “true”

// XST black box attribute definitions
// box_type “black_box”
// synthesis attribute box_type of instance_name
   is “black_box”

module logib1(A, DO, DI, WR_EN, WR_CLK);
// synthesis “black_box”
```

```
// Synplicity black box declaration
input [5:0] A;
output [3:0] DO;
input [3:0] DI;
input WR_EN;
input WR_CLK;
endmodule
```

3. Use **File** → **Open** on the Project Navigator menu to open the Verilog file in which the LogiBLOX component is to be instantiated.
4. Cut and paste the module declaration from the LogiBLOX component's .vei file into the Verilog design code, placing it after the *endmodule* line in the architecture section or the Verilog design code.
5. Cut and paste the component instantiation from the .vei file into the design code. Give the added code an instance name and edit it to connect the ports to the signals.

   Following is an example of Verilog source code with the LogiBLOX instantiation for the component named *logib1*. For each NGC file from LogiBLOX, you may have one or more Verilog files with the NGC file instantiated. In this example, there is only one black box instantiation of memory, but multiple calls to the same module may be done.

```
module top (D,CE,CLK,Q,
   Atop, DOtop, DItop, WR_ENtop, WR_CLKtop);

input D;
input CE;
input CLK;
output Q;

input  [5:0] Atop;
```

```
output  [3:0] DOtop;
input  [3:0] DItop;
input WR_ENtop;
input WR_CLKtop;

userff U0  (.D(D),.CE(CE),.CLK(CLK),.Q(Q));

memory U1  (   .A(Atop),
               .DO (DOtop),
               .DI (DItop),
               .WR_EN (WR_ENtop),
               .WR_CLK (WR_CLKtop));
endmodule
```

Note An alternate method is to place the module declaration from the .vei file into a new, empty Verilog file (MEMORY.V) and add the new file (shown below) to the project.

```
//----------------------------------------
// LogiBLOX SYNC_RAM Module “memory”
// Created by LogiBLOX version C.16
//    on Wed Jun 01 10:40:25 1999
// Attributes
//    MODTYPE = SYNC_RAM
//    BUS_WIDTH = 4
//    DEPTH = 48
//    STYLE = MAX_SPEED
//    USE_RPM = FALSE
//----------------------------------------
module MEMORY (A, DO, DI, WR_EN, WR_CLK);
input [5:0] A;
```

```
output [3:0] DO;
input [3:0] DI;
input WR_EN;
input WR_CLK;
endmodule
```

6. Save the Verilog source file. The design with the instantiated LogiBLOX module can now be checked for syntax errors and synthesized.

   Note When the design is synthesized, a warning is generated that the LogiBLOX module is unlinked. Modules instantiated as black boxes are not elaborated and optimized. The warning message is just reflecting the black box instantiation.

Simulating LogiBLOX Components

For simulation of LogiBLOX-created VHDL components, add the *component_name*.vhd file created by LogiBLOX to the project. This step is not necessary for Verilog components.

Constraining LogiBLOX Memory with FPGA Express

In the XSI (Xilinx Synopsys Interface) HDL methodology, whenever large blocks of memory are needed, LogiBLOX RAM or ROM modules are instantiated in the HDL code. With LogiBLOX RAM or ROM modules instantiated in the HDL code, you can specify in a UCF file the timing and placement constraints on these modules, and the primitives that comprise these modules.

To create timing or placement constraints for LogiBLOX memory modules, you must know:

- how many primitives will be used
- how the primitives and LogiBLOX modules are named

Estimating the Number of Primitives Used

When memory is specified with LogiBLOX, the memory depth and width are specified. If the RAM or ROM depth is divisible by 32, then 32x1 primitives are used. If the RAM or ROM depth is not divisible by 32, then 16x1 primitives are used instead. In the case of dual-port RAMs, 16x1 primitives are always used. Based on whether 32x1 or 16x1 primitives are used, the number of RAMs or ROMs can be calculated.

For example, if a RAM48x4 is required for a design, RAM16x1 primitives are used. Based on the width, there are four banks of RAM16x1s. Based on the depth, each bank has three RAM16x1s.

How the RAM Primitives are Named

Using the example of a RAM48x4, the RAM primitives inside the LogiBLOX are named as follows.:

- MEM0_0
- MEM1_0
- MEM2_0
- MEM3_0
- MEM0_1
- MEM1_1
- MEM2_1
- MEM3_1
- MEM0_2
- MEM1_2
- MEM2_2
- MEM3_2

Each primitive in a LogiBLOX memory module has an instance name of MEMx_y, where:

- *y* represents the primitive position in the bank of memory
- *x* represents the bit position of the memory output

For the next two items, see the Verilog and VHDL examples included at the end of this section. The Verilog and VHDL example instantiates a RAM32x2S, which is in the bottom of the hierarchy. The RAM32x2S is implemented with LogiBLOX. The next two items are written within the context of the Verilog examples but also apply to the VHDL examples as well.

Referencing a LogiBLOX Module or Component

This section discusses referencing a LogiBLOX module or component in an HDL source.

You constrain LogiBLOX memory modules in the HDL Flow using a UCF file. You can reference LogiBLOX memory modules instantiated in the HDL with the full-hierarchical instance name. If a LogiBLOX memory module is at the top level of the HDL code, the instance name of the LogiBLOX memory module is just the instantiated instance name.

In the case of LogiBLOX memory, which is instantiated within the hierarchy of the design, the instance name of the LogiBLOX module is the concatenation of all instances which contain the LogiBLOX memory. The concatenated instance names are separated by a "_". In the example, the RAM32X1S is named `memory`. The module `memory` is instantiated in the Verilog module `inside` with an instance name U0. The module `inside` is instantiated in the top-level module test. Therefore, you can reference the RAM32X1S in a UCF file as U0/U0. For example, to attach a TNM to this block of RAM, use the following line in the UCF file.

```
INST U0_U0 TNM=block1 ;
```

Because U0/U0 is composed of two primitives, a Timegroup called block1 is created; block1 TNM can be used throughout the UCF file as a Timespec end or start point, or U0/U0 can have a LOC area constraint applied to it. If the RAM32X1S is instantiated in the top-level file, and the instance name used in the instantiation is U0, this block of RAM can just be referenced by U0.

Referencing the Primitives of a LogiBLOX Module

This section discusses referencing the primitives of a LogiBLOX module in an HDL source.

Sometimes it is necessary to apply constraints to the primitives that compose the LogiBLOX memory module. For example, if you choose a floorplanning strategy to implement your design, it may be necessary to apply LOC constraints to one or more primitives inside a LogiBLOX memory module.

Returning to the RAM32x2S example above, suppose that each of the RAM primitives must be constrained to a particular CLB location. Based on the rules for determining the MEMx_y instance names and using the example from above, you can reference each of the RAM primitives by concatenating the full-hierarchical name to each of the MEMx_y names. The RAM32x2S created by LogiBLOX would have primitives named MEM0_0 and MEM1_0. So, for an HDL Flow project, CLB constraints in a UCF file for each of these two items would be:

```
INST U0_U0/MEM0_0 LOC=CLB_R10C10 ;
INST U0_U0/MEM0_1 LOC=CLB_R11C11 ;
```

Verilog Example

Following is a Verilog example.

test.v:

```
module test(DATA,DATAOUT,ADDR,C,ENB);

input [1:0] DATA;
output [1:0] DATAOUT;
input [4:0] ADDR;
input C;
input ENB;
wire [1:0] dataoutreg;
reg [1:0] datareg;
reg [1:0] DATAOUT;
reg [4:0] addrreg;
```

```
inside U0 (.MDATA(datareg),.MDATAOUT(dataoutreg),
           .MADDR(addrreg),.C(C),.WE(ENB));

always@(posedge C) datareg = DATA;
always@(posedge C) DATAOUT = dataoutreg;
always@(posedge C) addrreg = ADDR;

endmodule
```

inside.v:

```
module inside(MDATA,MDATAOUT,MADDR,C,WE);

input [1:0] MDATA;
output [1:0] MDATAOUT;
input [4:0] MADDR;
input C;
input WE;

memory U0 ( .A(MADDR), .DO(MDATAOUT),
            .DI(MDATA), .WR_EN(WE), .WR_CLK(C));

endmodule
```

test.ucf

```
INST “U0_U0” TNM = usermem;
TIMESPEC TS_6= FROM : FFS :TO: usermem: 50;
INST “U0_U0/mem0_0” LOC=CLB_R7C2;
```

VHDL Example

Following is a VHDL example.

test.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity test is
  port(
      DATA: in STD_LOGIC_VECTOR(1 downto 0);
      DATAOUT: out STD_LOGIC_VECTOR(1 downto 0);
      ADDR: in STD_LOGIC_VECTOR(4 downto 0);
      C, ENB: in STD_LOGIC);
   end test;

architecture details of test is
signal dataoutreg,datareg: STD_LOGIC_VECTOR(1 downto 0);
signal addrreg: STD_LOGIC_VECTOR(4 downto 0);

component inside
  port(
      MDATA: in STD_LOGIC_VECTOR(1 downto 0);
      MDATAOUT: out STD_LOGIC_VECTOR(1 downto 0);
      MADDR: in STD_LOGIC_VECTOR(4 downto 0);
      C,WE: in STD_LOGIC);
  end component;

begin

      U0: inside port
map(MDATA=>datareg.,MDATAOUT=>dataoutreg.,MADDR=>addrreg,C=>C,WE=>
ENB);

      process( C )
          begin
              if(Cevent and C=1) then
                  datareg <= DATA;
```

```
            end if;
        end process;

    process( C )
        begin
            if(Cevent and C=1) then
                DATAOUT <= dataoutreg;
            end if;
        end process;

    process( C )
        begin
            if(Cevent and C=1) then
                addrreg <= ADDR;
            end if;
        end process;

end details;
```

inside.vhd

```
entity inside is
  port(
      MDATA: in STD_LOGIC_VECTOR(1 downto 0);
      MDATAOUT: out STD_LOGIC_VECTOR(1 downto 0);
      MADDR: in STD_LOGIC_VECTOR(4 downto 0);
      C,WE: in STD_LOGIC);
  end inside;

architecture details of inside is component memory
  port(
      A: in STD_LOGIC_VECTOR(4 downto 0);
      DO: out STD_LOGIC_VECTOR(1 downto 0);
      DI: in STD_LOGIC_VECTOR(1 downto 0);
      WR_EN,WR_CLK: in STD_LOGIC);
  end component;

begin
      U0: memory port map(A=>MADDR,DO=>MDATAOUT,
              DI=>MDATA,WR_EN=>WE,WR_CLK=>C);
```

```
end details;
```

test.ucf

```
INST “U0_U0” TNM = usermem;
TIMESPEC TS_6= FROM : FFS :TO: usermem: 50;
INST “U0_U0/mem0_0” LOC=CLB_R7C2;
```

LogiBLOX Documentation

The *LogiBLOX Guide* is available with the Xilinx online book collection on the CD-ROM supplied with your software or from Xilinx at http://support.xilinx.com.

You can access LogiBLOX online help from LogiBLOX or from the Project Navigator online help system.

Chapter 9

CORE Generator

This chapter contains the following sections:

About CORE Generator

The Xilinx CORE Generator is a design tool that delivers parameterizeable cores optimized for Xilinx FPGAs. It provides a catalog of ready-made functions ranging in complexity from simple arithmetic operators such as adders, accumulators, and multipliers, to system-level building blocks including filters, transforms, memories.

Here are some of the enhancements in the integration between CORE Generator and ISE:

- Menu Selection Enhancements

  The menu selection **`File Save`** → **`Save Project As`** has been enhanced to copy any cores in the project as well as their associated files. This includes the `XCO`, `SYM`, `VHO`, `VEO`, `EDN`, and `MIF` files for each core. In addition, the CORE Generator project files are copied including `coregen`, `crp`, `coregen.pfj`, and `core.tpl`.

The menu selection **Project → Add Copy of Source** has also been enhanced to copy cores from other directories and their associated files. This includes the XCO, SYM, VHO, VEO, EDN, and MIF files for each core.

- New HDL flow improvements.

Design Flow

The CORE Generator System produces an Electronic Data Interchange Format (EDIF) netlist, and may also produce a schematic symbol, and/or a Verilog Output (VEO) with a Verilog (V) wrapper file, or VHDL Output (VHO) template file with a VHDL (VHD) wrapper file. The Electronic Data Netlist (EDN) file contains the information for implementing the module.

The schematic allows you to integrate the CORE Generator module into an ECS schematic. Finally, the VEO and VHO template files contain code that can be used as a model to instantiate a CORE Generator module in a Verilog or VHDL design so that it can be simulated and integrated into a design.

Note Verilog and VHDL wrapper files are used to pass information to the Xilinx CoreLib Simulation models. VHDL wrapper files are used only by ISE during simulation. Only Verilog wrapper files are used by ISE during Verilog synthesis to provide Port direction information to the Synthesizer.

You can download new cores from the Xilinx Web site and add these cores to the CORE Generator. The URL for downloading cores is http://www.xilinx.com/ipcenter. Check this Web site to verify that you have the latest version of each core and core data sheet.

Opening the CORE Generator Main GUI

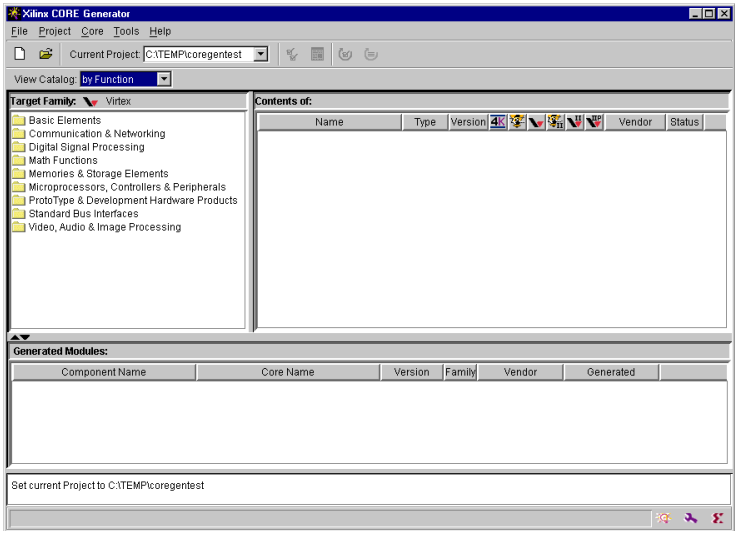

Figure 9-1 CORE Generator Main GUI (Sample)

In ISE 4, you can open the CORE Generator main GUI as a standalone program, or as an integrated design entry tool in the Project Navigator.

To open the CORE Generator as a standalone program, click **`Start`** → **`Programs`** →**`Xilinx`** → **`ISE 4`** →**`CORE Generator System`** from your PC's desktop.

The CORE Generator is an integrated design entry tool within ISE. To open the CORE Generator main GUI from Project Navigator:

1. Select a CORE Generator source by clicking it in the Sources in Project window.
2. Click the plus sign (+) to expand the COREgen toolbox in the Processes for Current Source window.
3. Double-click **`Manage Cores`**.

   The CORE Generator main window opens.

   Note This procedure assumes an existing core. To create a new core, see "Creating a CORE Component" below.

Creating a CORE Component

To create a CORE component:

1. Click **Project** → **New Source**.
2. Click **Coregen IP** from the New dialog box.
3. Enter a file name for the new core.
4. Verify that the information for the core (source name and source type) is correct.
5. Click **Finish** to launch the CORE Generator interface.
6. Click **Next**.
7. At the New Source Information window, click **Finish**.

   The CORE Generator Main Window opens.

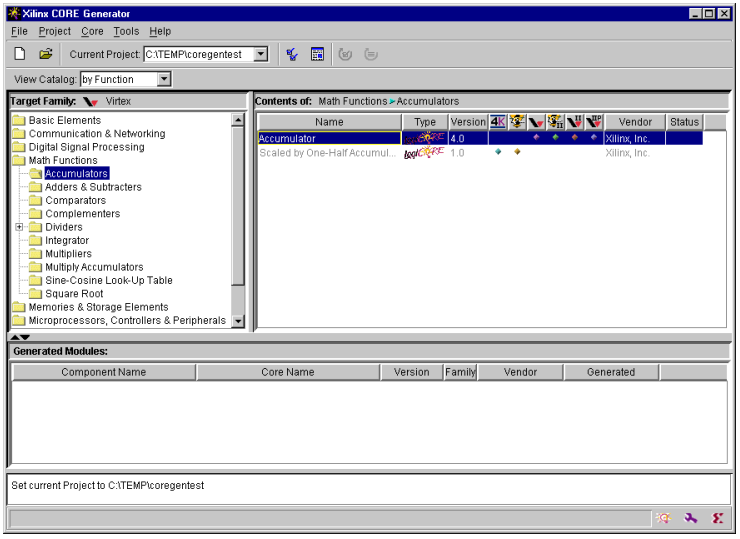

Figure 9-2 CORE Generator Main Window, Cores Selected

To access the project setup options, click **Project** → **Project Options** from the CORE Generator menu. However, if you opened the CORE Generator from an ISE 4 project, the Project Options are automatically set to the appropriate values for the project.

To select a core, double-click its name in the “Contents of” section of the CORE Generator window. A new window opens where you can

customize the core for your use, view its data sheet, and get other information concerning the core. The items that can be customized for a particular core depend on what the core is.

The next figure shows the CORE Generator window that opens when you select the accumulator core for a Virtex project.

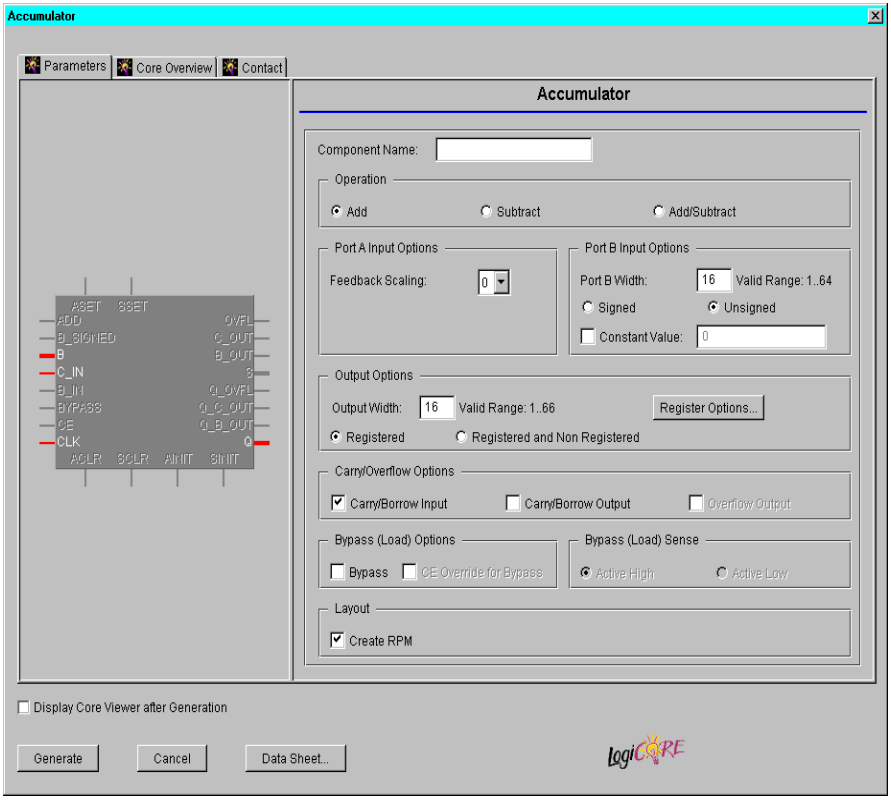

Figure 9-3 CORE Generator Window, Accumulator CORE

Note The Component Name field is blank on the CORE Generator customization screens.The name entered in the ISE 4 Project Navigator New window is not passed to the CORE Generator.

CORE Component Names

You must manually enter a name in the Component Name field. CORE component names have the following requirements:

- Must begin with a lower case alphabetic character a-z.
- No extensions or uppercase letters are allowed.
- After the first character, may include 0-9, _ (underscore).
- No HDL reserved words, for example:
  - Verilog - Do not use module, input, output.
  - VHDL - Do not use component, function, configuration, port, signal.

Click the Data Sheet button to view detailed information on the core. You must have the Adobe Acrobat Reader version 4.0 or above installed on your PC to view the data sheet.

After you customize the core, click the **`Generate`** button to generate the new core. When the core is successfully generated, the new core and its related files are placed in the project directory. When finished, close the CORE Generator windows.

The customized core component is a collection of several files including those listed below. The files are located in your ISE 4 project directory.

component_name.coe	ASCII data file defining the coefficient values for FIR filters and initialization values for memory modules
component_name.xco	CORE Generator file containing the parameters used to generate the customized core
component_name.edn	EDIF implementation netlist for the core
component_name.veo	Verilog template file
component_name.vho	VHDL template file
component_name.mif	Memory Initialization Module for Virtex Block RAM modules

component_name.vhd	VHDL wrapper file
component_name.v	Verilog wrapper file

The component name is the name given to the core in the customization window. The port names are the names provided in the template (.veo or .vho) files.

Using Cores in ISE 4 Projects

If you initiated the CORE Generator from the Project Navigator New window, the new core and its related files are placed in the current ISE 4 project directory. The related files include schematic, VHDL, and Verilog templates that can be used in your project. The core is added as a user document and appears in the Sources in Project window.

Editing Cores

To edit a core in an ISE 4 project, double-click its name in the Sources in Project window. This opens the CORE Generator's customization window for that core. After you make changes and click the Generate button, the files related to that core are updated and the ISE project is marked out of date.

Using Cores in Schematic Sources

The CORE Generator automatically creates a schematic symbol for any core component. When you create a core in an ISE 4 project, its schematic symbol is automatically added to the Engineering Capture System (ECS) local symbol library for use in schematic designs for that project.

To select a core for use in a schematic in an ISE 4 project:

1. Open the Engineering Capture System (ECS) either by double-clicking an existing schematic source in the Sources in Project window, or click **`Project`** → **`New source`**, then **`Schematic`** to create a new schematic source.
2. When the Engineering Capture System (ECS) is open, click **`Add`** → **`Symbol`**.
3. Select the core from the Local Symbol library for placement on the open schematic.

Instantiating Cores in an HDL Source

The CORE Generator automatically creates a VHDL template and Verilog template for each core component. When you create a core in an ISE 4 project, the HDL templates are automatically added to the Language Templates tool. These templates are then available for instantiation in VHDL or Verilog sources for the project.

To access the core component HDL instantiation templates for a project:

1. Click **Edit** → **Language Templates** from the Project Navigator menu.

   The Language Template window opens in the HDL Editor workspace.

2. Click the "+" icon beside **COREGEN** in the Language Templates window to access the directories under COREGEN.

3. For VHDL projects, click the "+" icon beside **VHDL Component Instantiation** directory to display the list of available VHDL instantiation templates. For Verilog projects, click the "+" icon beside **Verilog Component Instantiation**.

4. Double-click the core to display its instantiation template in the right pane of the Language Templates window.

5. To use the template, cut and paste the template text from the Language Templates window to a new or open HDL file.

Simulation and Synthesis of Core Modules

See the Schematic and HDL Design Flows chapter of the *CORE Generator Guide* accessed from the CORE Generator **Help** → **Online Documentation** for detailed information on general simulation and synthesis of CORE modules. See the "Synthesizing and Simulating Cores" section below for specific information on ISE and Coregen flows.

Synthesizing and Simulating Cores

This section discusses synthesizing and simulating cores with ISE integrated tools.

The CORE Generator now outputs VHDL or Verilog wrapper files when a core is created or modified. These wrapper files contain parameters important for simulation, including VHDL configuration statements. The wrapper methodology eliminates the necessity to specify configurations in your VHDL testbench because these are now fully contained within the wrapper file. Therefore, these files are necessary for correct behavioral simulation of CORE Generator cores.

In some cases, these wrapper files can be useful for synthesis as well. ISE will automatically use the wrapper file at the appropriate time. The table below is a list of flows and describes if the wrapper files are used by ISE in each flow. The simulation processes are those that are visible when a testbench is selected in the Sources In Project window.

Flows	Synthesis	Simulate Behavioral Model Process	Simulate Post-Translate, -MAP, and Place & Route Processes
All VHDL flows	Not Used	Used	Not Used (Not Necessary)
All Verilog Flows	Used (For module statement and port directions)	Used	Not Used (Not Necessary)

These wrapper files require the XilinxCoreLib library, so make sure that this library is mapped in your simulator project.

Chapter 10

Implementation

This chapter contains the following sections:

About Implementation

After you create a design source, the Implement Design processes convert the logical design represented in that source (and all sources in the hierarchy from that source down) into a physical file format that can be implemented in the selected target device. You can implement the design multiple times using different implementation process properties or target devices.

For detailed information about implementation in FPGAs and CPLDs, see the following chapters later in this Guide:

Implementing Design Processes

After design entry, use the processes in the Implement Design section of the Processes for Current Source window to:

- Implement the design for a specific target device
- Generate reports showing the status of the design
- Perform a timing analysis for design verification
- Export the design for timing simulation and programming

Note The Implement Design processes for your design depend on whether your design is for an FPGA or CPLD device. The synthesis tool selection does not affect the implementation flow.

Implementing Your Design

To implement your design:

1. Click a source file (HDL file, schematic, or EDIF file) in the Sources in Project window representing the design that you want to implement in the device. That source and all sources in the design hierarchy below it are included in the implementation processing.
2. Implementation processing is automatically customized for the targeted device. You can, however, try different processing options as necessary to meet your design requirements by setting processing properties. Right-click Implement Process and select Properties. See the "Properties" section of the "FPGA Implementation" chapter or the "Properties" section of the "CPLD Implementation" chapter for more information.

   Note Implementation process properties affect *all* sources in the project. They cannot be set for (or associated with) only certain sources.
3. Use the Implement Design processes in the Processes for Current Source window to select the implementation processing you want performed on the selected source (from that source down through the design hierarchy). You can implement your design completely, or request that only necessary processing be done to produce a specific implementation report.

Complete Implementation

For complete implementation processing, double-click **`Implement Design`** in the Processes for Current Source window. The Project Navigator checks the state of your project. It automatically initiates the processing necessary to completely implement the selected source. For example, it synthesizes your design, if necessary, before it starts the implementation process. When processing is complete, an icon indicates the results. A green check mark indicates that the design implemented without errors.

Partial Implementation

You can also request a partial implementation. For example, to view a report on the design only through a certain implementation stage, double-click the name of a report listed under the various processes in the Implement Design section of the Processes for Current Source window (**`Map Report`**, for example). If necessary, click the "+" box beside the Implement Design process to expand the Implement Design process list. If the report does not currently exist, the necessary processing is done to produce the report. If the report exists, it is opened immediately.

Specialized Processing

Click any of the Implement Design specialized processes to access them. These processes include: locking pins, multi-pass Place and Route, and launching advanced implementation tools. The User Constraint processes listed in the Design Entry Utilities section can be used for editing the UCF file or accessing the Constraints Editor GUI to add constraints to your design.

Implementation Errors and Warnings

As a source is implemented, the processes being run and their status are shown in the Transcript window. The processing status is also indicated by icons beside the process or report name in the Processes for Current Source window.

When processing is complete, an icon appears to the left of the Implement Design process, and to the left of any other processing it did.

- A green check mark indicates that the process was successfully completed.
- A red X indicates that errors were detected preventing successful completion of the process.
- A yellow exclamation point indicates a warning.

Saving Implementation Results

To preserve a specific version of the implementation:

- Archive a version of the project as a .zip file.
- Take a snapshot to preserve a project version for later reference and easy reactivation. See the "Snapshots and Archives" section of the "Project Navigator" chapter for detailed information on snapshots.

Deleting Results

To delete the implementation results of synthesis and implementation, click **Project→ Delete Implementation Data** from the Project Navigator menu. This function will delete all of the files and directories generated by the synthesis and implementation tools. In the case of an EDIF Design Flow, only the results of implementation are deleted.

Changing Devices

You may need to change device families to get the best results from your design. Implementation is automatically performed on the device set for the project in the Project Properties dialog box.

To change the device:

1. Double-click the Device and Design Flow line in the Sources in Project window.
2. In the Project Properties dialog box, select a new device or device family from the drop-down menus.

If you select a new device family, the implementation results (and synthesis results) become out-of-date. Click **Implement Design** or the report you want to view to implement the design with the new device.

Note If you change devices, you may need to use a different synthesis tool. See the "Creating a Project" section of the "Projects" chapter for detailed information on changing devices and synthesis tools.

Viewing Implementation Reports

The implementation reports provide information on:

- Logic trimming
- Logic optimization
- Timing constraint performance
- I/O pin assignment

Generating and Viewing a Report

To generate or view a report:

1. Click a design source in the Sources in Project window.
2. Double-click the report name in the Processes for Current Source window Implement Design section.
   - If the report currently exists (from previous processing), it is immediately opened. If the existing report is out-of-date (no green check mark beside it), click the report name, then click **`Process`** → **`Run`** (or **`Process`** → **`Rerun All`**) to update it before viewing it.
   - If the report does not currently exist, ISE 4.x runs the necessary processes to produce an up-to-date report and then displays it.

Note The reports produced depend on whether your design targets a CPLD or FPGA device.

Report Descriptions

For descriptions of each of the reports listed in the Implement Design section of the Processes for Current Source window, see the "FPGA Implementation" chapter and the "CPLD Implementation" chapter.

User Constraints

You can control the implementation of a design by defining constraints that affect the mapping and layout of the physical circuit. The User Constraints section under Design Entry Utilities in the Processes for Current Source window provides access to the mechanisms you can use to enter or modify constraints on the implementation of your design.

Editing the UCF File

The User Constraints File (UCF) is an ASCII file specifying constraints on the logical design. Enter constraints in the file with a text editor. These constraints affect how the logical design is implemented in the target device. The file can also be used to override constraints specified during design entry, earlier in the design flow.

A default UCF file is produced automatically when you create the project. This default UCF is named *top_source_name*.ucf where *project_name* is the name entered to create the project. To create or use a different UCF file, identify the UCF file in the Translate Options tab of the Implement Design Process Properties dialog box for FPGA designs (see the "FPGA Implementation" chapter). For CPLDs, the UCF file for the project is specified in the Design tab (see the "CPLD Implementation" chapter).

To edit the current UCF file:

1. Select a source in the Sources in Project window.
2. Double-click **`Edit Implementation Constraints File`** in the Design Entry Utilities section of the Processes for Current Source window.
3. The current UCF file opens in your text editor for viewing, printing, or modification.

See the *Xilinx Constraints Guide* for information on constraints that can be entered in the UCF file.

Whenever the UCF file is modified and saved, a message box reminds you that the implementation processes are not automatically updated when the UCF is changed.

Click **Reset** to have the implementation processing rerun with the new design source netlist and UCF file.

Opening the Xilinx Constraints Editor

Use the Xilinx Constraints Editor to place constraints (instructions) on symbols or nets in an FPGA or CPLD schematic or textual entry file such as VHDL or Verilog. Changes or additions to constraints through the Xilinx Constraints Editor are automatically added into the User Constraints File (UCF).

To open the Constraints Editor:

1. Click a source in the Sources in Project window.
2. Go to the Processes for Current Source window.
3. Click Design Entry Utilities.
4. Click User Constraints.
5. Double-click **Edit Implementation Constraints (Constraints Editor**).

   The Constraints Editor opens with the design loaded.

The UCF file is changed when constraints are added or edited through the Xilinx Constraints Editor. You must run the implementation process (click **Reset** in the Notice window that appears) using the new UCF file and design source netlist to have the Implement Design process reflect the change.

See the Constraints Editor online help for detailed information on using the Constraints Editor.

Chapter 11

FPGA Implementation

This chapter contains the following sections:

Flow

This section discusses FPGA implementation flow. The FPGA implementation flow contains the following steps:

The following figure shows the implementation portion of the Processes for Current Source window for a design that targets an FPGA device. The implementation processes are under "Implement Design."

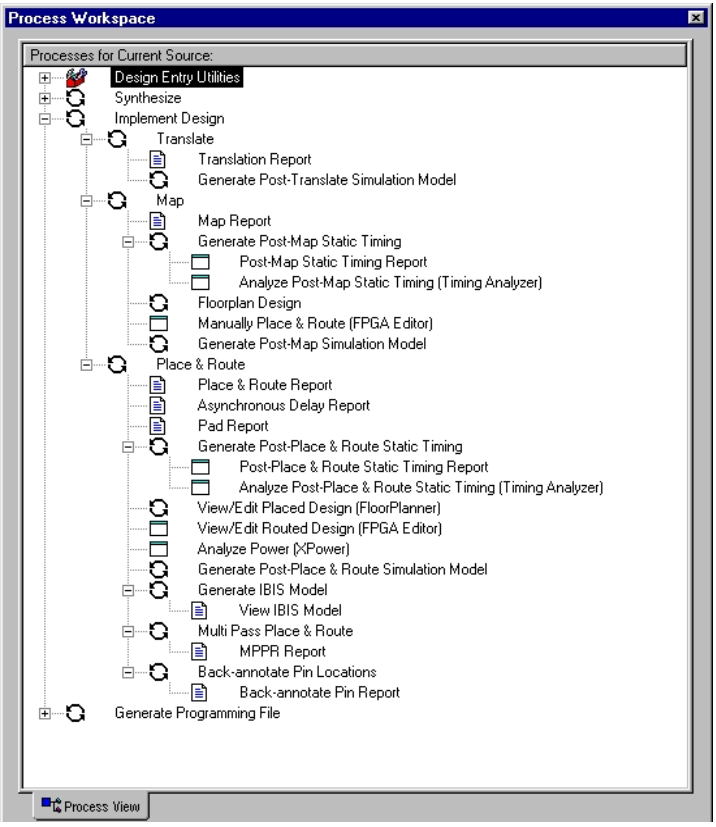

Figure 11-1 FPGA Implement Design Processes

Select a source and double-click **`Implement Design`** to perform the necessary processing to implement the design in the targeted device. Default implementation processing properties are used unless you modify them as described in the "Properties" section of this chapter.

Xilinx implementation tools are used to process your design. During the implementation, the design is converted from the logical design file format created in the design entry stage into a physical file format contained in an NCD (Native Circuit Description) file.

Implementation processing for FPGAs involves three basic phases: Translate, Map, and Place and Route. Processes to check and verify timing requirements are also included. Also included are processes to generate simulation models after each of these three implementation

phases. At the end of these phases, a programming file can be created. With the programming file you can optionally program a PROM or EPROM for subsequent programming of your Xilinx device, or directly download the programming file into the Xilinx device.

Translate

During the first step of design implementation, the translate process merges all of the input netlists and design constraint information and outputs a Xilinx NGD (Native Generic Database) file. The output NGD file can then be mapped to the targeted device family.

The following file types are *input files* for the translation process:

- Design file netlists (EDN, EDF, EDIF, or SEDIF files)
- User Constraints File (UCF File)

  The User Constraints File is an ASCII file that you create with a text editor or the Xilinx Constraints Editor. Its default name is *modulename*.ucf. The file contains timing and layout constraints that affect how the logical design is implemented in the target device. The constraints in the file are added to the information in the output NGD file.
- Netlist Constraints File (NCF file)

  The Netlist Constraints File (*modulename*.ncf) contains constraints specified within the Engineering Capture System (ECS) and synthesis tool. The netlist reader invoked by the Translate process adds the constraints to an intermediate NGO file and the output NGD file.
- Physical Macros (NMC files)

  The NMC files are binary files containing the implementation of a physical macro instantiated in the design. The Translate process reads the NMC file to create a behavioral simulation model for the macro.
- Module definition files (NGC files)

  NGC files are binary files containing the implementation of a module in the design. LogiBLOX creates an NGC file to define each module.

The following file types are *output files* for the translate process:

- Xilinx Native Generic Database (NGD file)

  The NGD file (*modulename*.ngd) is a binary file containing a logical description of the design expressed both in terms of the hierarchy used when the design was first created and in terms of lower-level Xilinx primitives to which the hierarchy resolves. The file also contains all of the constraints applied to the design during design entry or entered in a UCF (User Constraints File).

- Translation Report (BLD file)

  The Translation Report (*modulename*.bld file) contains information about the Translate (NGDBuild) run and its subprocesses. See the "Viewing Implementation Reports" section of the "Implementation" chapter for more information.

For a complete description of the Translate process, see the *Development System Reference Guide*.

MAP

The MAP process first performs a logical DRC (Design Rule Check) on the design in the NGD file produced by the Translate process. MAP then maps the logic to the components (logic cells, I/O cells, and other components) in the target Xilinx FPGA. The output design is an NCD (Native Circuit Description) file physically representing the design mapped to the components in the Xilinx FPGA. The NCD file can then be placed and routed.

The following file types are *input files* for the mapping process:

- Native Generic Database (NGD) file

  The Translate process outputs the NGD file (*modulename*.ngd) for mapping to the target device during the map process.

- Physical Macros (NMC files)

  The NMC files are binary files containing the implementation of a physical macro instantiated in the design. The Translate process reads the NMC file to create a behavioral simulation model for the macro.

- (Optional) Guide Design File (NCD file)

  A Guide Design File is an output NCD file from a previous MAP run that is used as an input to guide a later MAP run. You must identify the NCD file to be used in the Use Guide Design File (.ncd) option on the Map Properties tab of the implement Design Process properties dialog box (see the"Properties" section) before implementing the design.

- (Optional) Map Floorplanner File (MFP file) - Optional

  A Map Floorplanner File, previously generated by the Floorplanner, can be specified as an input file. The MFP file is essentially used as a guide file for mapping. You must identify the MFP file to be used in the Use Floorplanner File option on the Map Properties tab of the Implement Design Process properties dialog box (see the"Properties" section) before implementing the design.

- (Optional) MAP Directive File (MDF file) -- Optional

  The MAP Directive File (MDF) is an optional input file used for guided mapping. The MDF file describes how logic was decomposed when the guide design was mapped. MAP uses the hints in the MDF as a guide for logic decomposition in the guided mapping run.

The following file types are *output files* for the mapping process:

- Native Circuit Description (NCD file)

  The NCD file (*modulename_map*.ncd) contains a physical description of the design in terms of the components in the target Xilinx device.

- Physical Constraints File (PCF)

  The PCF file (*modulename*.pcf) is an ASCII text file containing the constraints specified during design entry expressed in terms of physical elements. The physical constraints in the PCF file are expressed in Xilinx's constraint language. See the *Xilinx Libraries Guide* for a description of this file.

- Native Generic Map File (NGM file)

  An NGM file (*modulename*.ngm) is a binary design file containing all of the data in the input NGD file as well as information on the physical design produced by the mapping. The NGM file is used

to correlate the back-annotated design netlist to the structure and naming of the source design.

- MAP Directive File (MDF file)

  The MDF file describes how logic was decomposed when the design was mapped. In guided mapping, MAP uses the hints in the MDF as a guide for logic decomposition.

- MAP Report (MRP file)

  The Map Report (*modulename*.mrp) contains information about the Map run and its subprocesses. See the "Viewing Implementation Reports" section of the "Implementation" chapter for more information.

For a detailed description of the MAP process, see the *Development System Reference Guide*.

Generate Post-Map Static Timing (Optional)

The Generate Post-Map Static Timing process (the Xilinx TRACE program) creates timing simulation data. Use this data to determine if the timing requirements and functionality of your design are correct before implementing the place and route process.

The following files are *input files* for the Pre-Route Static Timing process:

- A mapped design file (NCD file)

  The NCD file (*modulename_map*.ncd) contains a physical description of the design in terms of the components in the target Xilinx device.

- (Optional) Physical Constraints file (PCF file)

  The PCF file (*modulename*.pcf) is an ASCII text file containing the constraints specified during design entry expressed in terms of physical elements. The physical constraints in the PCF file are expressed in Xilinx's constraint language. See the *Xilinx Libraries Guide* for a description of this file.

  Constraints can indicate such things as clock speed for input signals, the external timing relationship between two or more signals, absolute maximum delay on a design path, or a general timing requirement for a class of pins.

The post-map static timing process outputs the Static Timing Report. The Static Timing Report (*modulename_preroute*.tw1) contains information about how well the timing constraints for the design have been met. See the "Viewing Implementation Reports" section of the "Implementation" chapter for more information.

For a complete description of the timing processes, see the *Development System Reference Guide*.

Place and Route

After an FPGA design has undergone the necessary processing to bring it into the mapped NCD format, it is ready to be placed and routed. This phase is done by PAR (Xilinx's Place and Route program). PAR takes a mapped NCD file, places and routes the design, and produces an NCD file to be used by the programming file generator (BitGen). The output NCD file can also act as a guide file when you place and route the design again after you make minor changes to it.

In the Xilinx Development System, PAR places and routes a design using a combination of two methods:

- *Cost-based* placement and routing are performed using various cost tables which assign weighted values to relevant factors such as constraints, length of connection and available routing resources.
- *Timing-driven* PAR places and routes a design based upon your timing constraints.

The following file types are *input files* for the Place and Route process:

- A mapped design file (NCD file)

  The mapped NCD file (*modulename_map*.ncd) contains a physical description of the design in terms of the components in the target Xilinx device.

- Physical Constraints File (PCF)

  The Physical Constraints File (*modulename*.pcf) is an ASCII file containing constraints specified during design entry expressed in terms of physical elements. The physical constraints in the PCF file are expressed in Xilinx's constraint language. See the *Xilinx Libraries Guide* for a description of this file.

- (Optional) Native Circuit Description Guide file (NCD file)

  You can optionally specify a template NCD file for placing and routing the design. You must identify the template file in the "Use Guide Design File" field of the Place and Route Options tab on the Implement Design Process Properties dialog box.

The following file types are *output files* for the Place and Route process:

- Placed and Routed Design File (NCD file)

  The NCD file (*modulename*.ncd) is a placed and routed design file. It may contain placement and routing information in varying degrees of completion. This file can be used by the programming file creation program (BitGen) to produce a bitstream file for downloading.

- Place-and-Route Report (PAR file)

  The Place-and-Route Report (*modulename*.par) includes summary information of all placement and routing iterations. See the "Viewing Implementation Reports" section of the "Implementation" chapter for more information.

- Asynchronous Delay Report (DLY file)

  The Delay File (*modulename*.dly) contains delay information for each net in the design. See the "Viewing Implementation Reports" section of the "Implementation" chapter for more information.

- Pad Report (PAD file)

  The PAD Report (*modulename*.pad) contains I/O pin assignments. See the "Viewing Implementation Reports" section of the "Implementation" chapter for more information.

- Guide Report File

  A Guide Report File is created if you identified a guide file in the "Use Guide Design File" field on the Place and Route Options tab.

- Intermediate Failing Timespec Summary (XPI file)

  The Intermediate Failing Timespec Summary is a report generated for failing timing specifications Xilinx Par Information.

This report appears regardless whether the design was routed and the timing specifications were met.

For a complete description of PAR, see the *Development System Reference Guide*.

Generate Post-Place & Route Timing (Optional)

The Post-Place & Route Timing process (the Xilinx TRACE program) can be run after Place and Route to create timing simulation data. Use this data to determine if the timing requirements and functionality of your design have been met after routing.

The following files are *input files* for the post-route timing process:

- A placed and routed design file (NCD file)

  The NCD file (*modulename*.ncd) contains a physical description of the design in terms of the components in the target Xilinx device.

- (Optional) Physical Constraints file (PCF file)

  The PCF file (*modulename*.pcf) is an ASCII text file containing the constraints specified during design entry expressed in terms of physical elements. The physical constraints in the PCF file are expressed in Xilinx's constraint language. See the Xilinx *Libraries Guide* for a description of this file.

  Constraints can indicate such things as clock speed for input signals, the external timing relationship between two or more signals, absolute maximum delay on a design path, or a general timing requirement for a class of pins.

Output from the post-route timing process is the Post-Route Timing Report (*modulename* .twx). The Post-Place & Route Static Timing Report contains information about how well the timing constraints for the design have been met. There are two different types of timing reports: the error report and verbose report.

For a complete description of the pre-Route Static Timing process, see the *Development System Reference Guide*.

Multi Pass Place and Route (Optional)

Running multiple place and route passes allows you to automatically generate multiple place and route solutions for a design to find the best possible placement for the design. Optimal placement results in

shorter routing delays and faster designs. The Multi-Pass Place and Route process offers both the ability to generate a determinate, repeatable solution by using the same starting point, or cost table. The Multi-Pass Place and Route process also has the ability to generate a range of unique solutions by using different cost tables.

Multiple iterations of placement and routing produce an output design file, a PAR file, a DLY file, and a PAD file for each iteration. When multiple iterations are run they are placed in mppr_results.dir under the project directory. When the Multi Pass Place and Route process is performed, PAR records a summary of all placement and routing iterations in one PAR file in the *mppr_results* directory, then places the output files (in NCD format) in that directory. The output NCD files are named based on the placer level, router level, and cost table used. For example, the NCD file mppr_results.dir/5_5_1.ncd indicates placer level 5, router level 5, cost table 1 were used.

The Multi-Pass Place and Route process scores each place and route pass and uses the score to determine the best passes to save. Scores are based on factors such as the number of unrouted nets, the delays on nets, and conformity to your timing constraints. Once this process is completed, the best result is determined by finding the lowest score and the corresponding design files are copied back to the project directory.

On average, design speed from an arbitrary multiple pass place and route varies plus or minus 5 to 10 percent from the median design speed across all cost tables. About 1/3 of the cost tables result in speeds within 5 percent of the median, 1/3 in the 5 to 10 percent range, and 1/3 in the 10 to 15 percent range. When comparing performance from the absolute worst cost table to the absolute best, a spread of 25 to 30 percent is possible.

Note Ranges are narrower for Virtex devices and higher for XC4000 devices.

Use the Multi-Pass Place & Route process in the following situations:

- You want to evaluate whether worse than expected results are due to a poor starting point or cost table.

  In this case, run multiple place and route passes at any time during the design cycle by running 3 or 4 cost tables.

- You made small changes to a design at the end of the design cycle and performance falls 5 to 10 percent as a result of these changes.

  In this case, run 5 to 10 cost tables.

Note Using the Multi-Pass Place & Route process is not recommended for every design iteration. If you need the top performing cost tables to meet design performance, modify your design to remove one or more logic levels.

To run Multi Pass Place & Route on an FPGA design:

1. Click a top-level source in the Sources in Project window.
2. Double-click **`Multi Pass Place & Route`** in the Processes for Current Source window.

You can elect to use the Multi Pass Place & Route defaults or set your own property values in the Multi Pass Place & Route Options dialog box.

Backannotate Pin Locations (Optional)

Each time you implement a design, a file is created which contains the pin locations and logical pad names information. For FPGAs, pin locations and logical pad names are read from a placed NCD file (*modulename*.ncd).

When you are ready to commit an FPGA design pinout, click the top-level source file in the Sources in Project window, then double-click the **`Backannotate Pin Locs`** process under **`Implement Design`** → **`Place-and-Route`**. If no conflicts are found, the pinout information stored in the .ncd file (FPGAs) is appended to the end of the User Constraint File for your design (*modulename*.ucf or the .ucf file specified in the Implement Design Process Properties dialog box). This pinout is applied to all subsequent design implementations that you run.

The Notice dialog box opens whenever the pin locking process is successful.

Whenever changes are made to a UCF file, implementation needs to rerun beginning with the translate process to incorporate the changes into the project. Click **`Reset`** in the Notice window if you want to rerun implementation to read to the new UCF file. However, if you want to keep the current implementation report files valid and available for viewing, and also keep any post-implementation

processes valid so that you can proceed without rerunning implementation, click **Retain**.

Backannotate Pin Locs Report

For FPGAs, double-click **Backannotate Pin Locs Report** in the Processes for Current Source window to generate and view this report. The Backannotate Pin Locs Report (*modulename*.lck for FPGAs) is displayed in the report window.

The Backannotate Pin Locs Report has two sections: Constraint Conflicts Information and List of Errors and Warnings.

The Constraints Conflicts Information section does not display if there are fatal input errors; for example, missing inputs or invalid inputs. However, the created report file contains the List of Errors and Warnings. The Constraints Conflicts Information section has two subsections: Net name conflicts on the pin and Pin name conflicts on the nets. If there are no conflicting constraints, both subsections under the Constraint Conflicts Information section contain a single line indicating that there are no conflicts.

The List of Errors and Warnings displays only if there are errors or warnings.

Pin Loc Constraints in the UCF

Pin locking constraints are written to a PINLOCK section in the UCF file. The PINLOCK section begins with the statement #PINLOCK BEGIN and ends with the statement #PINLOCK END. By default, conflicting constraints are not written to the UCF file. Before creating a PINLOCK section in the UCF, if the conflicting constraints are discovered, this information is reported.

User-specified pin locking constraints are never overwritten in a UCF file. However, if the user-specified constraints are exact matches of pin-locked generated constraints, a pound sign (#) is added in front of all matching user-specified location constraint statements. The pound sign indicates that a statement is a comment. To restore the original UCF file (the file without the PINLOCK section), remove the PINLOCK section and delete the pound sign from each of the user-specified statements.

The pin locking process does not check if existing constraints in the UCF file are valid pin locking constraints. Comments inside an

existing PINLOCK section are never preserved by a new run of the pin locking process. If the pin locking process finds a CSTTRANS comment, it equates “INST name” to “NET name”, then checks for comments.

The pin locking process writes to an existing UCF file under the following conditions:

- The contents in the PINLOCK section are all pin lock matches and there are no conflicts between the PINLOCK section and the rest of the UCF file.
- The PINLOCK section contents are all comments and there are no conflicts outside the PINLOCK section.
- There is no PINLOCK section and no other conflicts in the UCF file.

Reports

This section briefly summarizes FPGA implementation reports. *For detailed information about the reports, see the Project Navigator online help.* The following reports provide information on the implementation processing of your FPGA design:

- Translation Report
  The Translation Report contains warning and error messages from the three translation processes: conversion of the EDIF or XNF style netlist to the Xilinx NGD netlist format, timing specification checks, and logical design rule checks.
- MAP Report
  The MAP report contains warning and error messages detailing logic optimization and problems in mapping logic to physical resources.
- Place & Route Reports
  Place & Route Reports include:
  - PAR Report
  - PAD Report
  - Asynchronous Delay Report

- Post-Map Static Timing Report
  The Post-Map Static Timing Report gives you a calculated worst-case timing for all signal paths in your design. It optionally includes a complete listing of all delays on each individual path in the design. It does not include insertion of stimulus vectors. The FPGA design must be mapped and can be partially or completely placed, routed, or both.
- Post-Place & Route Static Timing Report
  The Post-Place & Route Static Timing Report process gives you a calculated worst-case timing for all signal paths in your design. It optionally includes a complete listing of all delays on each individual path in the design.

Properties

This section briefly summarizes FPGA implementation properties. *For detailed information about implementation properties, see the Project Navigator online help.* You can set multiple properties to control the implementation processes for the design. For FPGAs, the implementation process properties specify how a design is translated, mapped, placed, and routed. Reporting options are also available. Open the Process Properties dialog box to modify implementation options.

Opening the Process Properties Dialog Box

To open the Process Properties dialog box:

1. Click a design source file in the Sources in Project window for a project that targets an FPGA device.

   Note Implementation properties are set for the whole design, not for the selected module file only.

2. Right-click **`Implement Design`** in the Processes for Current Source window.
3. Click **`Properties`** from the pull-down menu.

   The Process Properties dialog box for the Implement Design processes opens. See the following figure.

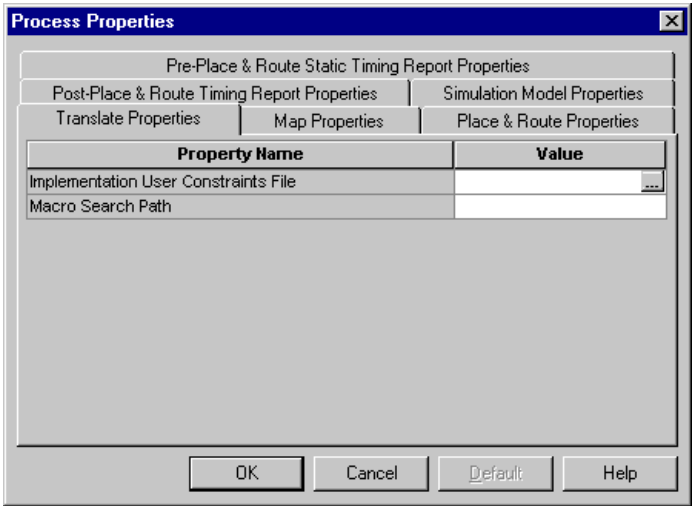

Figure 11-2 Sample Process Properties Dialog Box - Standard Display

Note This figure is a sample only. The actual content of the dialog box may vary depending on the type of process you are running, whether you have selected Standard or Advanced display, and other factors.

4. Click the tab corresponding to the type of options you want to set to display the available properties. Click **F1** to display detailed descriptions of the options on each tab.

   Note You can also right-click the processes listed under the Implement Design process to display the Process Properties dialog box for that process only

You can customize whether you want to display the Standard or Advanced list of properties in the Process Properties dialog boxes. Use the procedure described in the following section to display the Advanced properties.

Accessing Advanced Properties

The options listed in the Process Properties dialog boxes depend on whether you are using the Standard or Advanced display level. By default, only the Standard options are listed on each Process Property tab.

To access the Advanced options, do the following before you access the Implement Design Process Properties dialog box:

1. Click **Edit** → **Preferences** from the Project Navigator menu.
2. Click the Processes tab in the Preference dialog box.
3. Click **Advanced** for the Property Display Level.

Note The set of advanced properties is a true superset of standard properties. Accordingly, all the standard properties are also shown in advanced mode.

Tools

This section discusses FPGA Implementation Tools. The FPGA implementation tools include:

- "Floorplanner"
- "FPGA Editor"
- "Timing Analyzer"
- "XPower"

Processes available from these tools include:

- Analyze Post-Map Static Timing (Timing Analyzer)
- Floorplan Design (Floorplanner)
- Manually Place & Route (FPGA Editor)
- Analyze Post-Place & Route Static Timing (Timing Analyzer)
- View / Edit Placed Design (Floorplanner)
- View / Edit Routed Design (FPGA Editor)
- Analyze Power (XPower)

For detailed information on these processes, see the online help for the respective tool.

Floorplanner

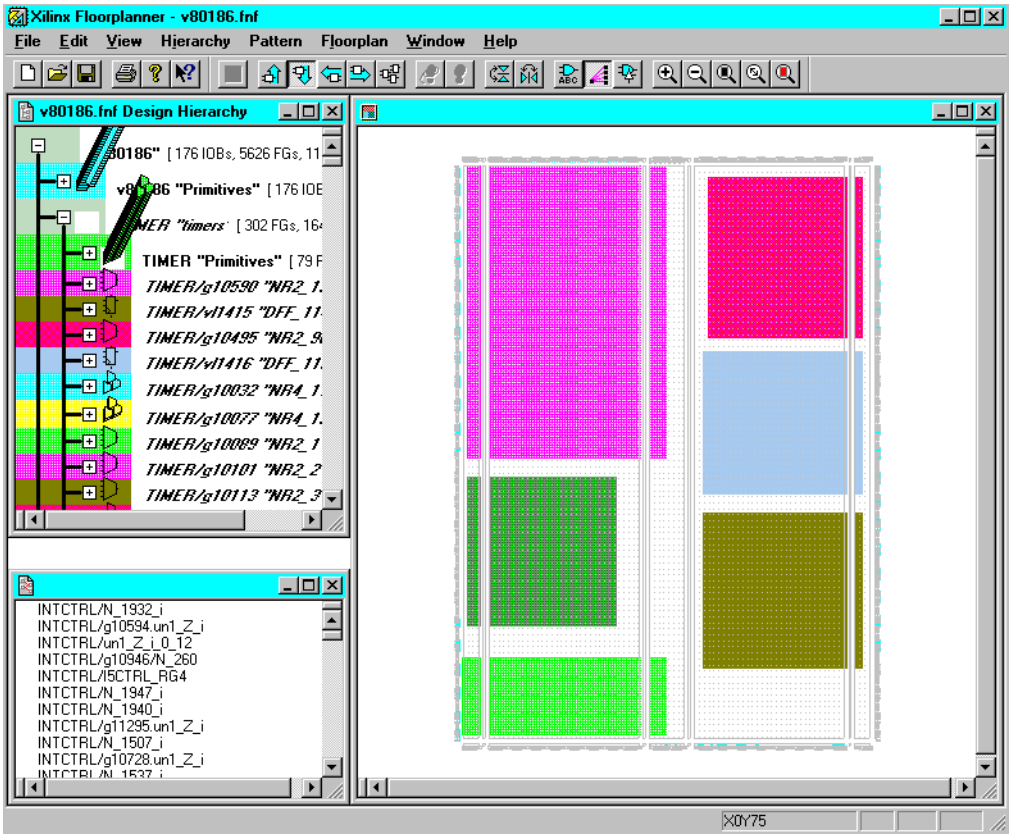

Figure 11-3 Floorplanner

The Floorplanner is a graphical placement tool that gives you control over placing a design into a target FPGA. You drag and drop elements of your design into the floorplan of the target device displayed in the Floorplanner window. The Floorplanner displays a hierarchical representation of the design in its Design Hierarchy window using hierarchy structure lines and colors to distinguish the different hierarchical levels.

To open the Floorplanner:

1. Click a design source in the Sources in Project window.
2. Double-click **Floorplanner** in the Processes for Current Source window.

   The Floorplanner opens and automatically loads the project's NGD file.

Floorplanning is particularly useful on structured designs and data path logic. With the Floorplanner, you see where to place logic in the floorplan for optimal results, placing data paths exactly at the desired location on the die.

You can use the Floorplanner at multiple points in the design implementation flow. The Floorplanner can be used:

- Prior to Mapping
- Prior to Place and Route
- After Place and Route

An NCD file is no longer a prerequisite for using the Floorplanner, and you can now output a UCF file. Invoking the Floorplanner after the design has been placed and routed allows you to view and possibly improve the results of the automatic implementation. In an interactive floorplan design flow, you floorplan and place and route, interactively. You can modify the logic placement in the Floorplan window as often as necessary to achieve your design goals. You can save the iterations of your floorplanned design to use later as a constraints file (MFP file) for MAP.

With the Floorplanner, you can floorplan your design before or after Place and Route. Invoking the Floorplanner after the design has been placed and routed allows you to view and possibly improve the results of the automatic implementation. In an iterative floorplan design flow, you floorplan and place and route, interactively. You can modify the logic placement in the Floorplan window as often as necessary to achieve your design goals. You can save the iterations of your floorplanned design to use later as a constraints file (MFP file) for PAR.

The Floorplanner generates an MFP file that contains mapping and placement information. You can use this file as a guide for mapping an implementation revision for FPGA devices.

For detailed information on using the Floorplanner, see the Floorplanner online help.

FPGA Editor

The FPGA Editor is a graphical application for displaying and configuring FPGAs. The FPGA Editor provides a graphic view of your placed and routed FPGA design, allowing you to make modifications. Use the FPGA Editor to place and route critical components before running the automatic place and route tools on your designs. You can also use the FPGA Editor to manually finish placement and routing if the routing program does not completely route your design. In addition, the FPGA Editor reads from and writes to the Physical Constraints File (PCF).

To open the FPGA Editor:

1. Click a design source in the Sources in Project window.
2. Double-click **FPGA Editor** in the Processes for Current Source window.

   The FPGA Editor opens and automatically loads the project's NCD file.

The following figure shows an example of the FPGA Editor.

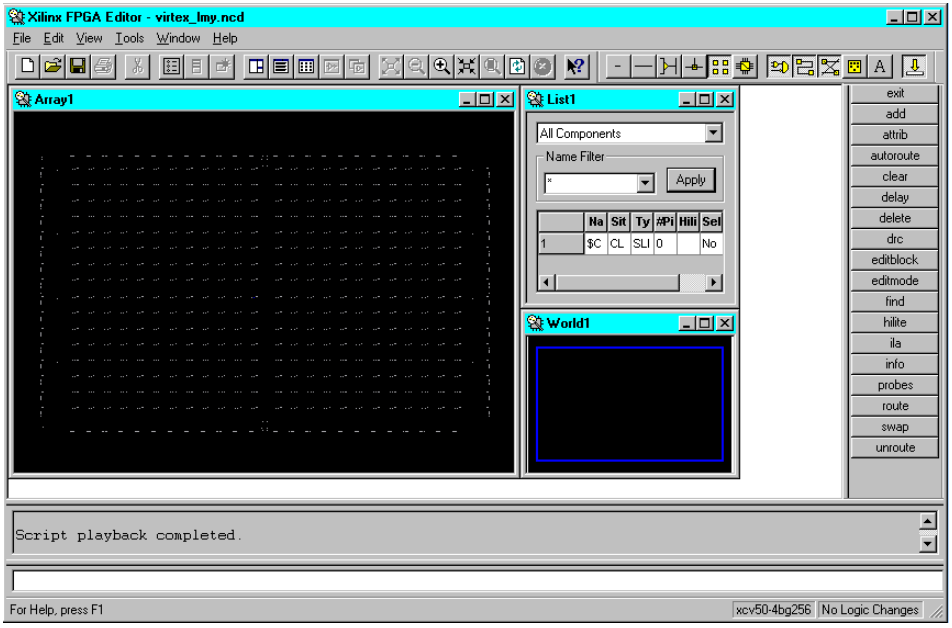

Figure 11-4 FPGA Editor Main Window

For detailed information on using the FPGA Editor, see the FPGA Editor online help.

Timing Analyzer

Use the Timing Analyzer program to perform static timing analysis on an FPGA or CPLD design. The Timing Analyzer:

- Verifies that the delay along a given path or paths meets your specified timing requirements.
- Creates timing analysis reports that you customize by applying filters.
- Organizes and displays data that allows you to analyze the critical paths in your circuit, the cycle time of the circuit, the delay along any specified paths, and the paths with the greatest delay.
- Provides a quick analysis of the effect of different speed grades on the same design.

To open the Timing Analyzer:

1. Click a design source in the Sources in Project window.
2. Double-click **`Timing Analyzer`** in the Processes for Current Source window.

   The Timing Analyzer opens and automatically loads the project's NGD file.

The FPGA design must be mapped and can be partially or completely placed, routed or both. The CPLD design must be completely placed and routed. A static timing analysis is a point-to-point analysis of a design network. It does not include insertion of stimulus vectors.

The Timing Analyzer works with synchronous systems composed of flip-flops and combinatorial logic. In synchronous design, the Timing Analyzer takes into account all path delays, including clock-to-Q and setup requirements, while calculating the worst-case timing of the design. However, the Timing Analyzer does not perform setup and hold checks; you must use a simulation tool to perform these checks.

For a complete description of the Timing Analyzer, see the Timing Analyzer's online help.

XPower

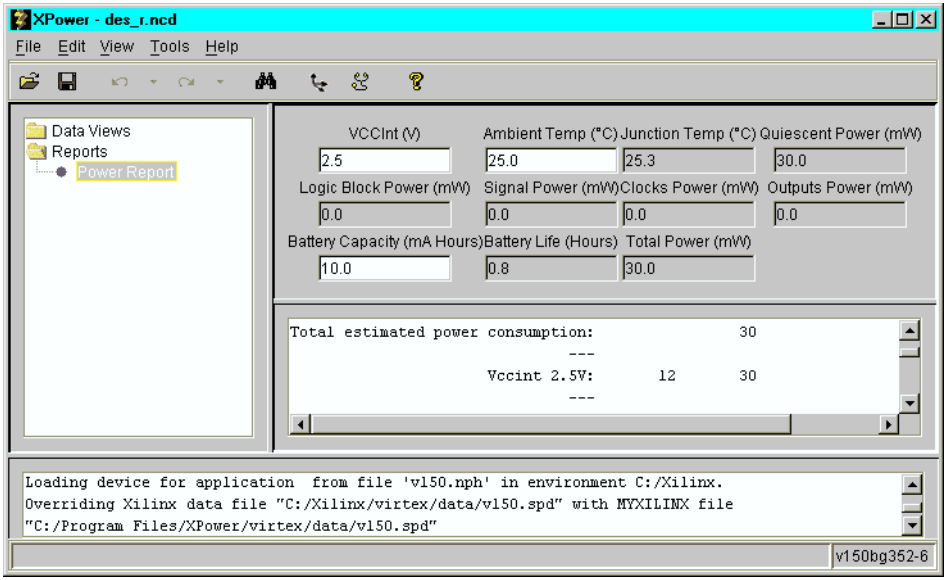

Figure 11-5 XPower Main Window

XPower is a graphical user tool that gives you power consumption information for specific designs. The XPower Summary Bar displays a representation of total estimated power consumed by a specific device.

Note The following sections are from the *XPower Release Notes*. See those notes for additional information. For in-depth information on how to use XPower, see its online help.

Why Estimate Power?

Power estimation is an optional methodology to help you estimate the power consumption of a digital design. It allows you to determine if your design will meet your power constraints. It also gives you detailed information about how much power specific components of a design will consume. Power estimation is particularly useful for power sensitive designs.

With XPower, you can estimate the power consumption of your design after running PAR. You may then identify power hungry parts of the design and make changes according to power constraints. In an

iterative design flow, one can power estimate and Floorplan, interactively. You can modify the power consumption in XPower as often as necessary to achieve your design goals. This may include, but is not limited to: reducing clock rates, reducing loads on FPGA outputs, redesigning critical sections of the design to consume less power, or reusing sections of the design for multiple purposes. XPower allows you to calculate power consumption of a particular design file based on frequency, toggle rate, enable rate, voltage, and load assignments. In the XPower application, power calculations are displayed on screen.

XPower Prerequisites

XPower is specifically intended to assist those users who require some degree of handcrafting for their designs. You must understand both the details of the device architectures and how power estimation can be used to refine a design. Successful power estimation is very much an iterative process and it can take time to develop a power level that outperforms an automatically processed design.

Because of the nature of XPower's interaction with the automatic MAP and PAR tools, several prerequisites are necessary in order to power estimate your design successfully.

- Detailed knowledge of the specifics of the target architecture and part
- Detailed knowledge of the specifics of the design being implemented
- A willingness to iterate power consumption to achieve the desired results
- Realistic performance and density goals

Features of XPower

XPower provides an easy-to-use graphical interface that offers the following features:

- Interacts at a high level of the design hierarchy, as well as with low-level elements such as I/Os, function generators, 3-state buffers, flip-flops, and RAM/ROM
- The Outputs view provides output power for OBUFS from both clocked and unclocked logic.

- Estimates power consumption for a design before it is downloaded.
- Considers the design resource usage, toggle rates, Input/Output power, and many other factors in estimation.
- Formulae used for calculations in the application are based on the classic power equation $P=CFV^2$, where P=power, C=capacitance, V=voltage and F=frequency.
- Because XPower is an estimation tool, results may not precisely match actual power consumption.
- In the absence of simulation information, the user is required to enter a clock frequency and estimated toggle rate percentage to be applied to all the signals in each path. The toggle rate will be a default value unless changed by the user. Even when toggle rates are available, clock paths are still a valid organization of the data, allowing the user to identify paths that consume a large amount of power. Clock enable signals are used to divide all the logic in a clock path into smaller collections of logic. This will help the user assign appropriate toggle rates for sections of logic and help them identify logic paths that may have power problems.

Chapter 12

CPLD Implementation

This chapter contains the following sections:

Implement Design

This section discusses CPLD implementation flow, which includes:

The following figure shows the implementation portion of the Processes for Current Source window for a design that targets a CPLD device. The implementation processes are under "Implement Design."

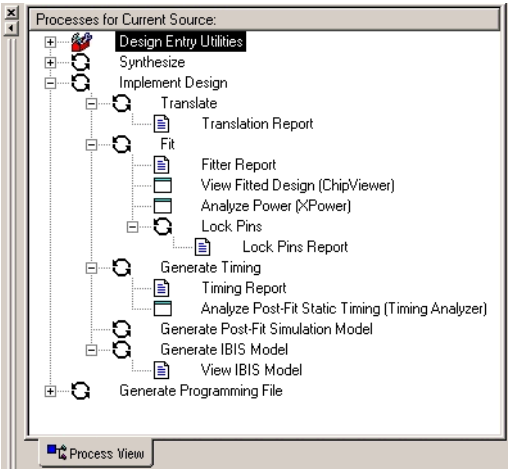

Figure 12-1 CPLD Implement Design Processes

When you select a source and click **`Implement Design`**, the necessary processing to implement the design in the targeted CPLD device is performed. Default implementation processing properties are used unless you modify them as described in the "Properties" section.

During implementation, the design is converted from the logical design file format created in the design entry stage into a physical file format contained in a JED file. Implementation processing for CPLDs involves two basic phases: Translate and Fit.

Translate

This section contains the following topics:

- "About the Translate Process"
- "Translation Report"

About the Translate Process

CPLD designs go through the same translate process that FPGA designs do. During the first step of design implementation, the translate process merges all of the input netlists and design constraint information and outputs a Xilinx NGD (Native Generic Database)

file. The output NGD file can then be mapped to the targeted device family.

The following file types are *input files* for the translate process:

- Design file EDIF netlists (EDN files)
- User Constraints File (UCF File)

  The User Constraints File (default name is *project_name*.ucf) is an ASCII file that you create with a text editor or using the Xilinx Constraints Editor. The file contains timing and layout constraints that affect how the logical design is implemented in the target device. The constraints in the file are added to the information in the output NGD file.
- Netlist Constraints File (NCF file)

  The Netlist Constraints File (*top_source_name*.ncf) contains constraints specified within the Engineering Capture System (ECS) and synthesis tool. The netlist reader invoked by the Translate process adds the constraints to an intermediate NGO file and the output NGD file.
- Module definition files (NGC files)

  NGC files are binary files containing the implementation of a module in the design. Some compilers like XST create an NGC file to represent a design module together with its constraints.

The following file types are *output files* for the translate process:

- Xilinx Native Generic Database (NGD file)

  The NGD file (*top_source_name*.ngd) is a binary file containing a logical description of the design expressed both in terms of the hierarchy used when the design was first created and in terms of lower-level Xilinx primitives to which the hierarchy resolves. The file also contains all of the constraints applied to the design during design entry or entered in a UCF (User Constraints File).
- Translation Report (BLD file)

  The Translation Report (*top_source_name*.bld file) contains information about the Translate (NGDBuild) run and its subprocesses. See the "Viewing Implementation Reports" section of the "Implementation" chapter and the "Translation Report" section below for more information.

For a complete description of the Translate process, see the *Development System Reference Guide*.

Translation Report

The Translation Report (*top_source_name*.bld) contains warning and error messages from the three translation processes: conversion of the EDIF netlist to the Xilinx NGD netlist format, timing specification checks, and logical design rule checks. The report lists the following:

- Missing or untranslatable hierarchical blocks
- Invalid or incomplete timing constraints
- Output contention, loadless outputs, and sourceless inputs

Fit

This section includes the following topics:

- "About the Fit Process"
- "Fitter Report"
- "View Fitted Design (ChipViewer)"
- "Analyze Power (XPower)"
- "Lock Pins"

About the Fit Process

The NGD file output by the Translate process is input to the Fit process. During Fit, the CPLD Fitter minimizes and collapses the combinatorial logic of your design so that it requires the least number of macrocell and product term resources. It also partitions and maps your design to fit within the architecture of the CPLD. You can control aspects of this step by setting implementation options using properties included on the Basic tab in the CPLD Implement Design Process Properties dialog box.

Fit Process Output Files

The following file types are *output files* for the Fit process:

- Fitter Report (RPT file)

  The Fitter Report (*top_source_name*.rpt) shows information such as the type and quantity of device resources used and the resulting pinout. See the "Viewing Implementation Reports" section of the "Implementation" chapter and the "Fitter Report" section below for more information.

- Guide file (GYD file)

  The Guide file (*top_source_name*.gyd) contains all resulting pinout information. The pinout information stored in the Guide file is copied into the User Constraint File (UCF) when the "Lock Pins" process is run. During subsequent design iterations, the information copied into the UCF is written only upon successful completion of the fitter.

- Design Database File (VM6 file)

  The Design Database file (*top_source_name*.vm6) is a binary file containing the physical mapping of your design into the target CPLD resulting from the Fitter process.

For detailed information about implementing CPLD designs, see *CPLD Design Flows* tutorial in the online help (**`Help`** → **`ISE Help Contents`** from the Project Navigator).

Delete Implementation Data

The feature Delete Implementation Data deletes the following files:

- deleting file(s): __projnav.log
- deleting file(s): _chipview.pl
- deleting file(s): ngdbuild.rsp _"filename"_sch2vhf_exewrap.rsp _ngdbld.rsp __"filename"_2prj_exewrap.rsp
- __filesAllClean_exewrap.rsp
- deleting file(s): "filename".vhf
- deleting file(s): xst
- deleting file(s): "filename".prj

- deleting file(s): “filename”.xst
- deleting file(s): “filename”.syr
- deleting file(s): “filename”.cup
- deleting file(s): “filename”._prj
- deleting file(s): _cpldfit.tcl

For specific information, see the topic “Cleanup Project Files” in the ISE online help.

Fitter Report

The Fitter Report *(top_source_nam*e.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.

View Fitted Design (ChipViewer)

This process runs the ChipViewer. The View Fitted Design (ChipViewer) process provides a graphical view of the CPLD fitting report. With this tool you can examine inputs and outputs, macrocell details, equations, and pin assignments. See the“View Fitted Design (ChipViewer)” section below.

Analyze Power (XPower)

This process runs XPower. XPower is a graphical user tool that gives you power consumption information for specific designs. See the “XPower” section of the “FPGA Implementation” chapter for more information about using XPower to estimate power consumption.

The CPLD fitter also provides a CXT file used by the XPower software tool. The CXT file contains all of a specific CPLD design's architectural elements related to power consumption. The CXT file is used by XPower to estimate the power consumed by a specific design. It is recommended to use a VCD (Value Change Dump) file which is exported from a simulation using ModelSimXE (TM) to facilitate accurate power estimation.

Lock Pins

Lock Pins is optional. Each time you implement a design, a file is created which contains the pin locations and logical pad names information. For CPLDs, this information is saved in a Guide file (*top_source_name*.gyd).

When you are ready to commit a CPLD design pinout:

1. Click the top-level source file in the Sources in Project window.
2. Double-click the **Lock Pins** process under **Implement Design** → **Fitter**.
3. If no conflicts are found, the pinout information stored in the .gyd file (CPLDs) is appended to the end of the User Constraint File for your design (*top_source_name*.ucf or the .ucf file specified in the Implement Design Process Properties dialog box). This pinout is applied to all subsequent design implementations that you run.

For CPLDs, you can use an external guide file to lock pins. To specify the location of the external guide file:

1. Right-click the **Lock Pins** process.
2. Click **Properties**.
3. Enter the file path in the **Use External GYD File** field in the Process Properties dialog box.
4. Click **OK**.
5. Double-click **Lock Pins** to run that process with the specified GYD file.

A Notice information window appears whenever the pin locking process is successful.

Whenever changes are made to a UCF file, implementation needs to rerun beginning with the translate process to incorporate the changes into the project. Click **Reset** in the Notice window if you want to rerun implementation to read to a new UCF file. However, if you want to keep the current implementation report files valid and available for viewing and also keep any post-implementation processes valid so that you can proceed without rerunning implementation, click **Retain**.

ChipViewer can be used to graphically enter pin locking constraints into the UCF file. Using this method, you need not know the specific UCF file syntax.

If you originally implemented your design targeting a device selected automatically by the Fitter ("AUTO" in the targeted device name), you should change your targeted device selection to the specific device and package combination selected by the fitter when you are ready to lock your pinout. Pinout constraints may become invalid if you continue to target an AUTO device and the fitter chooses a different device or package.

Lock Pins Report

After running the Lock Pins process, the report file can be viewed by double-clicking on the Lock Pins Report process. The report will show any pin assignment conflicts that may have occurred. The Lock Pins Report is named *top_source_name*_lcr.

Pin Loc Constraints in the UCF

For both FPGAs and CPLDs, pin locking constraints are written to a PINLOCK section in the UCF file. The PINLOCK section begins with the statement #PINLOCK BEGIN and ends with the statement #PINLOCK END. By default, conflicting constraints are not written to the UCF file. Before creating a PINLOCK section in the UCF, if the conflicting constraints are discovered, this information is reported.

User-specified pin locking constraints are never overwritten in a UCF file. However, if the user-specified constraints are exact matches of pin-locked generated constraints, a pound sign (#) is added in front of all matching user-specified location constraint statements. The pound sign indicates that a statement is a comment. To restore the original UCF file (the file without the PINLOCK section), remove the PINLOCK section and delete the pound sign from each of the user-specified statements.

The pin locking process does not check if existing constraints in the UCF file are valid pin locking constraints. Comments inside an existing PINLOCK section are never preserved by a new run of the pin locking process. If the pin locking process finds a CSTTRANS comment, it equates "INST name" to "NET name", then checks for comments.

The pin locking process writes to an existing UCF file under the following conditions:

- The contents in the PINLOCK section are all pin lock matches and there are no conflicts between the PINLOCK section and the rest of the UCF file.
- The PINLOCK section contents are all comments and there are no conflicts outside the PINLOCK section.
- There is no PINLOCK section and no other conflicts in the UCF file.

Constraint Conflicts Information

Following is a sample pin2ucf Report file:

```
pin2ucf Report File
Copyright (c) 1995-2001 Xilinx, Inc.  All rights
   reserved.
           Created : Fri Jun 29 11:01:10 2001

Constraint Conflicts Information
================================

   This section provides information on the
   constraint conflicts
   if pin2ucf were to write a ucf file using the
   provided design and
   the existing ucf file. There are 2 types of
   conflicts that can
   occur.

  1. Multiple pins could be constrained on the same
   net
   2. Same pin could have multiple nets

   pin2ucf provides a list each for both.

  Note:- “New NET” and “New PIN” indicates the Net
   and Pin
   locations found suitable by pin2ucf for pin
   locking, while
```

```
   "Old Net" and "Old PIN" indicates the Net and Pin
    locations
    already set by the user in the ucf file, which
    were left
    untouched by pin2ucf.


Net name conflicts on the pins
-------------------------------

-------------------------------------------------
   -------------------
PIN Location                      New NET
   Old NET

-------------------------------------------------
   -------------------
"No net name conflicts were found on pins"

Pin name conflicts on the nets
-------------------------------
-------------------------------------------------
   -------------------
NET Name            New PIN         Old PIN
-------------------------------------------------
   -------------------
"No pin name conflicts were found on nets"

PinLocking Successful. Constraints written to UCF
   File
```

Generate Timing

You can run the Timing process to verify that your design meets your timing requirements. A timing report is generated with input constraint timing violations.

Timing Report

The Timing Report Format property under the Timing process selects between a Summary report (default) and a Detailed report. The Summary report provides the net worst-case timing between all pairs

of connected points (pads and registers) in the design. The Detailed report shows a breakdown of all internal path delay parameters comprising each point-to-point path in the design.

The Timing Report *(top_source_nam*e.tim) shows a summary report of worst-case timing for all paths in the design. It optionally includes a complete listing of all delays on each individual path in the design.

Analyze Post-Fit Static Timing (Timing Analyzer)

This process runs the Timing Analyzer. The Timing Analyzer performs static timing analysis of an FPGA or CPLD design. The CPLD design must be completely placed and routed. A static timing analysis is a point-to-point analysis of a design network. It does not include insertion of stimulus vectors. See the "Timing Analyzer" section below and the Timing Analyzer online help for detailed information.

Generate Post-Fit Simulation Model

This process runs synthesis and ngdbuild, map and fit to generate a NCD file. The NCD file is then used to generate a NGA file using NGDANNO. The NGA file is then used to generate the Simulation Model using NGD2VHDL or NDGD2VER.

To generate a Post-Fit Simulation Model:

1. Select the appropriate source file (ABEL) from the Sources in Project window.

   Note In a VHDL flow, only a VHDL model can be generated. In a Verilog flow, only a Verilog model can be created.

2. Select the Generate Post-Fit Simulation Model from the Processes for Current Source window and double-click.

Generate IBIS Model

IBISWriter is a batch tool that can be run from the GUI or the command line. It takes a Xilinx physical design (.ncd) file as input and writes an IBIS (.ibs) file as output. An IBIS file produced by IBISWriter primarily consists of a list of pins used by the design, the signals internal to the device that connect to those pins, and the IBIS buffer model that applies to the IOB that is connected to that pin. IBIS (Input Output Buffer Information Specification) provides Input/

Output device characteristics through V/I data (buffer models). It does not disclose any circuit/process information. It is a behavioral modeling specification suitable for transmission line simulation of digital systems. It is applicable to most digital components.

Properties

This section discusses CPLD Implementation Properties. You can set multiple properties to control the implementation process for the design. For CPLDs, they control how a design is translated and fit. Implementation options are specified in the Process Properties dialog box.

Note For detailed information on CPLD Implementation Properties, see the Project Navigator online help.

Opening the Process Properties Dialog Box

To set CPLD implementation properties:

1. Click a design source file in the Sources in Project window for a project that targets a CPLD device.

   Note Implementation properties are set for the whole design, not for the selected source file only.

2. Right-click **`Implement Design`** in the Processes for Current Source window.

3. Click **`Properties`** from the pull-down menu for Implement Design.

   The Process Properties dialog box for Implement Design opens.

4. Click a tab to access the list of properties you can set for the implementation options.

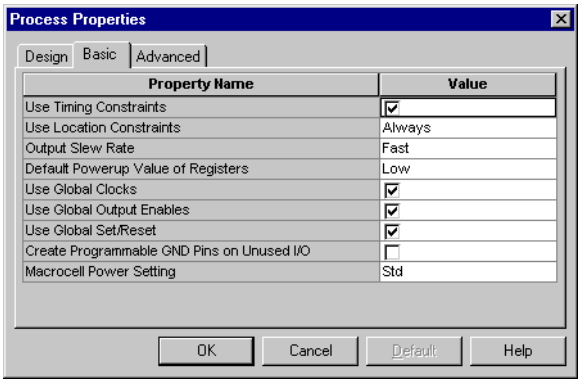

Figure 12-2 Sample Process Properties Dialog Box (CPLD)

The options listed in the CPLD Implement Design Process Properties dialog boxes are the same regardless of whether you are using the Standard or Advanced display level. The Display Level is set in the Processes tab of the Preferences dialog box (**`Edit`** → **`Preferences`** from the Project Navigator menu).

Setting Options

You can set the following options:

- "Translate Options"
- "Fit Options"
- "Lock Pins Options"
- "Timing Options"

Translate Options

To open a Process Properties dialog box with options specific to the Translate process:

1. Click a top-level source in the Sources in Project window.
2. Right-click **`Translate`** in the Processes for Current Source window.
3. Click **`Properties`** from the pull-down menu.
4. The Design tab displays with only translation-related properties.

See the "Setting Options" section for information on the User Constraints File option.

Fit Options

To open a Process Properties dialog box with options specific to the Fit process:

1. Click a top-level source in the Sources in Project window.
2. Right-click **Fit** in the Processes for Current Source window.
3. Click **Properties** from the pull-down menu.

   The Implement Design Properties dialog box opens.

Lock Pins Options

To open the Lock Pins Process Properties dialog box:

1. Click a top-level source in the Sources in Project window.
2. Right-click **Lock Pins** in the Processes for Current Source window.
3. Click **Properties** from the pull-down menu.

   The Lock Pins Process Properties dialog box opens.
4. In the Value field, enter the file path of the external guide file to use with the Lock Pins process. See the "Fit" section for information on the Lock Pins process.

Timing Options

To open the Timing Process Properties dialog box:

1. Click a top-level source in the Sources in Project window.
2. Right-click **Timing** in the Processes for Current Source window.
3. Click **Properties** from the pull-down menu.

   The Timing Report Process Properties dialog box opens.

The Timing Report Format property allows you to select the level of detail in the Timing Report. By default, a Summary report is produced containing summary timing information and design statistics. You can set the value to Detail to have the Timing Report include timing delay information for all nets and paths.

Tools

This section discusses CPLD Implementation Tools. Advanced implementation tools are available under the Launch Tools section of the Implement Design process. These tools are specifically intended to assist those users who require some degree of handcrafting for their designs. You must understand both the details of the device architectures and how the tool can be used to refine a design.

The CPLD implementation tools include:

- "Timing Analyzer"
- "CPLD ChipViewer"

Timing Analyzer

Timing Analyzer provides a powerful, flexible, yet easy way to perform static timing analysis on your design. Analysis may be performed immediately after mapping, or after placing and routing the design.

Timing Analyzer may be controlled through GUI features or its comprehensive macro facility. The user can analyze the same design against different physical constraint (PCF) files. Timing Reports have a hierarchical browser to quickly jump to different sections of reports. Timing paths in reports can be crossprobed to Synthesis tools (Exemplar and Synplicity) and Floorplanner. Timing Analyzer uses Timing Wizard for all FPGA analysis functions, ensuring consistency with PAR.

You can use the Timing Analyzer program to perform static timing analysis on both FPGA and CPLD designs. The Timing Analyzer is used to verify that the delay along a given path or paths meets your specified timing requirements. It creates timing analysis reports that you customize by applying filters. It organizes and displays data that allows you to analyze the critical paths in your circuit, the cycle time of the circuit, the delay along any specified paths, and the paths with the greatest delay. It also provides a quick analysis of the effect of different speed grades on the same design.

While an FPGA design must be mapped and can be partially or completely placed, routed or both, a CPLD design must be completely placed and routed. A static timing analysis is a point-to-

point analysis of a design network. It does not include insertion of stimulus vectors.

The Timing Analyzer works with synchronous systems composed of flip-flops and combinatorial logic. In synchronous design, the Timing Analyzer takes into account all path delays, including clock-to-Q and setup requirements, while calculating the worst-case timing of the design. However, the Timing Analyzer does not perform setup and hold checks; you must use a simulation tool to perform these checks.

To open the Timing Analyzer:

1. Click a design source in the Sources in Project window.
2. Double-click **`Timing Analyzer`** in the Processes for Current Source window.

   The Timing Analyzer opens and automatically loads the project's NGD file.

For a complete description of the Timing Analyzer, see the Timing Analyzer online help.

CPLD ChipViewer

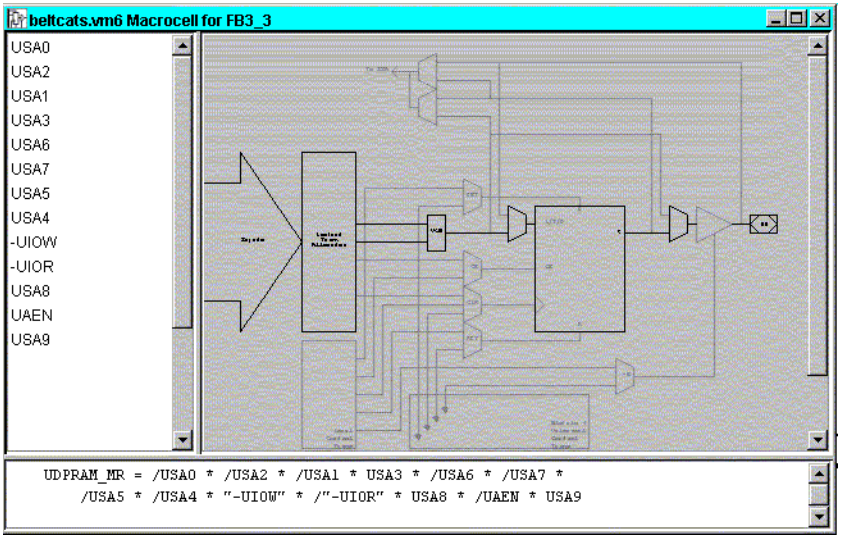

Figure 12-3 CPLD ChipViewer

The ChipViewer provides a graphical view of the CPLD fitting report. With this tool you can examine inputs and outputs, macrocell details, equations, and pin assignments.

For Pin Assignments

Before running the Fitter, you can use the ChipViewer to control pin assignments for implementation. To open the ChipViewer for pin assignment:

1. Click the top-level design source in the Sources in Project window.
2. Double-click **Pin Assignment, ChipViewer** in the User Constraint section under Design Entry Utilities.

   Note This command provides a graphical way to enter pin locking constraints for your design.

   The ChipViewer opens with the complete netlisted source design loaded.

The Pin Assignment ChipViewer process is available only if you have a specific CPLD device and package selected for your project.

Note This process is available under **Design Entry Utilities** → **User Constraints** → **Assign Pins (ChipViewer)**. To use this feature, you must select a specific device. Auto device selection is not supported with this feature.

View Fitted Design (ChipViewer)

After implementation is complete, you can use the ChipViewer to examine the physical mapping resulting from the Fitter. To open the ChipViewer to examine Fitter results:

1. Click a top-level design source.
2. Double-click **View Fitted Design, ChipViewer** in the Launch Tools section under Implement Design.

   Note This command provides a graphical view of a subset of the CPLD Fitter report, plus additional information on the fitted design not available in the report. You can view pin placement, slew rates, macrocell power modes, specific pin-to-pin timing, and the connections between macrocells and pins.

The ChipViewer opens with the completely mapped design loaded.

For more information on using the CPLD ChipViewer, see its online help.

Chapter 13

Device Programming

This chapter contains the following sections:

When the design meets your requirements, the final step is to program the target device.

Creating FPGA Programming Files

After the design has been completely routed, you must configure the device so that it can execute the desired function. Xilinx's bitstream generation program, BitGen, takes a fully routed NCD (Native Circuit Description) file as its input and produces a configuration bitstream—a binary file with a .bit extension.

The BIT file contains all of the configuration information from the NCD file defining the internal logic and interconnections of the FPGA, plus device-specific information from other files associated with the target device. The binary data in the BIT file can then be downloaded into the FPGA's memory cells, or it can be used to create a PROM file.

To create a configuration bitstream file:

1. Click the top-level source for the project in the Sources in Project window.
2. Click **`Generate Programming File`** in the Processes for Current Source window.
3. Click **`Process`** → **`Run`** in the Project Navigator menu.

The programming file creation process runs. If there are no errors, the *top_source_name*.bit file is created.

4. To view the Programming File Report in the report window, double-click **`View Programming File Generation Report`** in the Processes for Current Source window.

   The Programming File Report contains information about the BitGen run.

For a complete description of BitGen, see the *Development System Reference Guide*.

Launching Programming Tools

Select a programming tool to configure the target device. See the "Device Programming Tools" section below for a short overview of each tool.

To launch a programming tool:

1. Click the top-level source file in the Sources in Project window.
2. Double-click the programming tool name in the Processes for Current Source window.

   The selected programming tool opens in its own window with the bitstream file loaded.

Setting FPGA Programming File Creation Options

You can set before programming file creation options before creating the programming file. To open the Process Properties dialog box containing these options:

1. Click the top-level design source file in the Sources in Project window for a project that targets an FPGA device.
2. Right-click **`Generate Programming File`** in the Processes for Current Source window.
3. Click **`Properties`** from the pull-down menu.

   The Process Properties dialog box for the Generate Programming File process appears.

4. Click the tab for the options you want to set. You can set properties for the following options:

   - General Options
   - Configuration Options
   - Startup Options
   - Readback Options

   These options are described in detail in the Project Navigator online help.

   Note You can specify whether to display the Standard or Advanced list of properties in the Process Properties dialog boxes.

Generating CPLD Programming Files

At the end of a successful CPLD implementation, a design database file (*top_source_name*.vm6) is created. From this, a JEDEC programming file can be generated. The iMPACT configuration tool uses this JED file to configure XC9500/XL/XV and XPLA3 (Coolrunner) CPLD devices.

Note For more information, see *CPLD Design Flow Tutorials*, accessible from the Project Navigator online help.

To create a JED programming file:

1. Click the top-level source for the project in the Sources in Project window.
2. Click **`Generate Programming File`** in the Processes for Current Source window.
3. Click **`Process`** → **`Run`** in the Project Navigator menu.

   The programming file creation process runs. If there are no errors, the *top_source_name*.jed file is created.

Setting CPLD Programming File Creation Options

This section describes the programming options you can set before creating the programming file. To open the Process Properties dialog box containing these options:

1. Click the top-level design source file in the Sources in Project window for a project that targets a CPLD device.
2. Right-click **`Generate Programming File`** in the Processes for Current Source window.
3. Click **`Properties`** from the pull-down menu.
4. The Process Properties dialog box for the Generate Programming File process appears.

You can set the following options for CPLD programming file creation:

- Signature / User Code

  Values: Blank (default) / Four-character text string

  Enters a unique text string in the Value field to identify the configuration 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 iMPACT configuration tool to identify the configuration data signature (usercode) of a programmed XC9500 or XC9500XL device. The default is to use the *top_source_name*.

- Jedec Test Vector File

  Includes a TMV file in your JEDEC file. The TMV file is a test vector file generated when ABEL compiles a design containing user test vectors.

- Disable BUSHOLD Circuitry (9500XL / XV Only)
  Disables bus-hold keepers for all device I / Os.

Device Programming Tools

After creating the programming file, use one of the following programming tools to configure your device:

- "PROM File Formatter"
- "iMPACT"

PROM File Formatter

An FPGA or daisy chain of FPGAs can be configured from serial or parallel PROMs. The PROM File Formatter can create MCS, EXO, or TEK style files. The files are read by a PROM programmer that turns the image into a PROM.

A HEX file can also be used to configure an FPGA or a daisy chain of FPGAs through a microprocessor. The file is sorted as a data structure in the microprocessor boot-up code.

The PROM File Formatter is available for FPGA designs only. The PROM File Formatter provides a graphical user interface that allows you to:

- Format BIT files into a PROM file compatible with Xilinx and third-party PROM programmers
- Concatenate multiple bitstreams into a single PROM file for daisy chain applications
- Store several applications in the same PROM file

iMPACT

The iMPACT configuration tool, a command line and GUI based tool, allows you to configure your PLD designs using Boundary-Scan, Slave Serial, and Select MAP configuration modes. iMPACT supports both the Parallel (JTAG) and MultiLINX cables.

iMPACT also allows you to:

- ReadBack and Verify design configuration data
- Run Boundary-Scan TAP debug operations
- Create SVF and STAPL Files

See the *iMPACT User Guide* and the iMPACT online help for details.