Foundation Series ISE 3.1i User Guide

Foundation Series ISE 3.1i User Guide — Online *Printed in U.S.A.*

About This Manual

The Xilinx Foundation Series Integrated Synthesis Environment (ISE) 3.1i consists of an integrated tool suite that helps you produce, test, and implement designs for Xilinx FPGAs or CPLDs. The available tools cover all aspects of the work flow from design entry to bitstream generation and downloading. You can directly access the internet from many of its applications for user support and tool updates. This *Foundation Series ISE 3.1i User Guide* provides a detailed description of the ISE design environment and tools.

Note This Xilinx software release is certified as Year 2000 compliant.

Manual Contents

This guide contents the following chapters:

- "Introduction"
- "Design Environment"
- "Creating a Project"
- "Project Navigator"
- "HDL Sources"
- "Schematic Sources"
- "State Diagrams"
- "LogiBLOX"
- "CORE Generator"
- "HDL Library Mapping"

- "Design Constraints / UCF File"
- "Simulation"
- "Synthesis"
- "Implementing the Design"
- "Snapshots"
- "Programming the Device"

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/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 contain device-specific information on Xilinx device characteristics, including readback, boundary scan, configuration, length count, and debugging `http://support.xilinx.com/partinfo/databook.htm`
Xcell Journals	Quarterly journals for Xilinx programmable logic users `http://support.xilinx.com/xcell/xcell.htm`
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 each convention.

Typographical

The following conventions are used for all documents.

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

  ```
  speed grade: - 100
  ```

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

  ```
  rpt_del_net=
  ```

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

  `File` → **`Open`**

- *Italic font* denotes the following items.

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

    `edif2ngd` *`design_name`*

  - References to other manuals

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

- Emphasis in text

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

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

  edif2ngd *[option_name] design_name*

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

  lowpwr ={on|off}

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

  lowpwr ={on|off}

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

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

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

  `allow block` *block_name loc1 loc2locn;*

Online Document

The following conventions are used for online documents.

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

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

Contents

Chapter 3 Creating a Project

Chapter 4 Project Navigator

Chapter 5 HDL Sources

Chapter 6 Schematic Sources

Chapter 11 Design Constraints/UCF File

Chapter 12 Simulation

Chapter 13 Synthesis

Chapter 14 Implementing the Design

Chapter 1

Introduction

This chapter contains the following sections:

- "Platform Support"
- "Xilinx Architecture Support"
- "Partner Tools"
- "Foundation Series ISE Demo"
- "Tutorials"
- "Online Help"
- "Books"

Platform Support

Foundation Series ISE 3.1i is designed for use on PCs running Windows 98, Windows NT 4.0 with SP4, SP5, or SP6, and Windows 2000.

Foundation Series ISE is supported on the following networks:

- Windows NT shares (server or workstation)
- PC NFS (e.g., Hummingbird and Intergraph/Maestro)
- Samba on Unix Servers

Note Foundation Series ISE is not supported on Novell networks.

Xilinx Architecture Support

The software supports the following Xilinx architecture families in this release.

- Spartan™/XL/-II

- Virtex™ / -E / -II
- XC9500™ / XL / XV
- XC4000™E / L / EX / XL / XLA

Note The devices available for your design depend upon the product configuration you purchased and on the synthesis tool you select for your project.

Partner Tools

Foundation Series ISE 3.1i includes the partner tools listed below to provide a complete design environment.

- Synopsys® : FPGAExpress

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

- Visual Software Solutions (VSS): StateCAD® , StateBench™, HDL Bencher™

  StateCAD and StateBench add state diagram creation and testing as a design entry option for Foundation Series ISE. The HDL Bencher provides automatic testbench / test fixture creation.

- Model Technology Incorporated (MTI): ModelSIM™

  ModelSIM simulators provide the simulation functions for Foundation Series ISE.

Foundation Series ISE Demo

You can run a QuickStart presentation after you install Foundation Series ISE. To run the demo, select **`Start`** → **`Programs`** → **`Foundation Series ISE v3.1i`** → **`Documentation`** → **`Multimedia QuickStart Presentation`**.

Tutorials

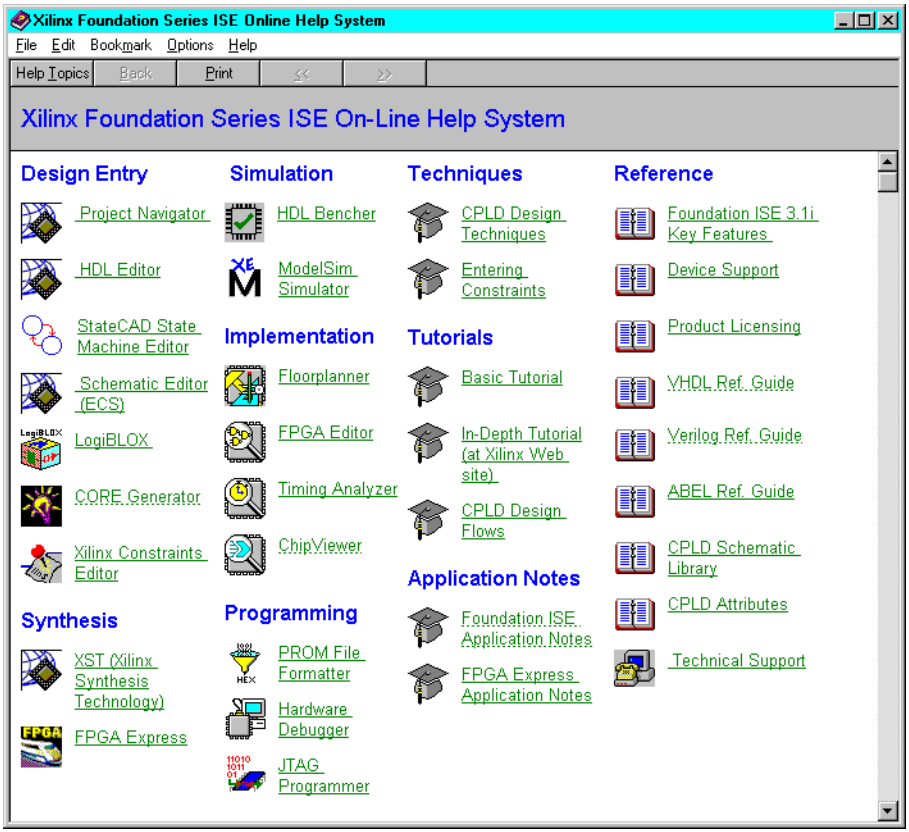
The *Foundation Series ISE 3.1i Quick Start Guide* contains a basic tutorial that describes the creation of a 4-bit counter module. The tutorial includes creating HDL and schematic source files for the design, functionally simulating the design's logic, processing the design for

device implementation, and using basic timing simulation to test the design in the device.

An in-depth tutorial, the Watch tutorial, is available from the Education tab on the Xilinx support website (`http://support.xilinx.com/support/techsup/tutorials/index.htm`.)

Online Help

Context sensitive help (press **F1**) is available for all Foundation Series ISE's tools. In addition, you can access Foundation Series ISE's umbrella help by selecting **Help** → **Foundation ISE Help Contents**. The umbrella help menu is shown in the following figure.



Books

Multiple printed and online books are available for the tools included and integrated with Foundation Series ISE 3.1i.

Printed Books

The *Foundation Series ISE 3.1i Installation Guide and Release Notes* describes installation procedures, key features, supported devices, and the most critical known issues.

The *Foundation Series ISE 3.1i Quick Start Guide* provides an overview of the key features of the Foundation Series ISE software. It also contains a tutorial that demonstrates the basic design process.

PDF Files

Adobe Acrobat PDF files are included in this release for viewing and printing all of the books in the Foundation Series ISE 3.1i online book collection. They can be found on the Documentation CD or accessed from the Xilinx support page on the web at `http://support.xilinx.com`.

Online Book Collection

The online book collection for Foundation Series ISE is available on the Foundation Series ISE 3.1i Documentation CD or from the Xilinx support page on the web at `http://support.xilinx.com`. You must use a Java-enabled HTML browser to the view the Xilinx online books.

Document Viewer

The Document Viewer provided with Foundation Series ISE 3.1i is powered by the Docsan™ indexing tool. This tool provides your HTML browser with optimal searching capabilities within the online book collection. Refer to the online help provided with the Document Viewer for detailed instructions on using this tool.

Xilinx Foundation Series ISE 3.1i Book List

The following table identifies and describes the books in the Foundation Series ISE online book collection. These are available from the Documentation CD or from the Web.

Table 1-1 Foundation Series ISE 3.1i Online Book Collection

Title	Description
General Information:	
Foundation Series ISE 3.1i Quick Start Guide	This guide gives an overview of Foundation Series ISE's tools and features. It includes a tutorial that demonstrates the basic Foundation Series ISE design process.
Foundation Series ISE 3.1i User Guide	This guide provides a detailed description of the Foundation Series ISE design environment.
Design Entry Online Reference Books:	
Libraries Guide	This book describes the logic elements (primitives or macros), that you use to create your designs as well as the attributes and constraints used to process elements during logic implementation. It also discusses relationally placed macros (RPMs). RPMs are macros that contain relative location constraints (RLOC) information. The Xilinx libraries enable you to convert designs easily from one family to another.
LogiBLOX Guide	This guide describes the high-level modules you can use to speed up design entry and the attributes that support logic synthesis, primarily for XC4000, Spartan, SpartanXL, and CPLD architectures. It also explains how to use the LogiBLOX program to create designs and the different types of logic synthesis completed by the LogiBLOX program.
StateCAD Reference Guide	(PDF file only) This book provides details on using the StateCAD and StateBench tools provided by VSS. (NOTE: This book is on the ALLSTAR VSS CD only.)

Table 1-1 Foundation Series ISE 3.1i Online Book Collection

Title	Description
Synthesis and Simulation Reference Books:	
Synthesis and Simulation Design Guide	This manual provides a general overview of designing FPGAs with Hardware Description Languages (HDLs). It includes design hints for the novice HDL user, as well as for the experienced user who is designing FPGAs for the first time.
ModeSIM XE User Guide	(PDF file only.) This guide describes using the Xilinx Version of MTI's ModelSIM. (NOTE: This book is on the MTI CD only.)
Xilinx Synthesis Technology (XST) User Guide	This guide describes XST support for Xilinx devices, HDL languages, and design constraints. The manual explains how to use various design optimization and coding techniques when creating designs for use with XST.
(Synopsys) *Verilog Reference Guide*	(PDF file only) This guide describes doing Verilog designs and synthesizing them with Synopsis' FPGA Express. (Can be accessed from **Help** → **Foundation ISE Help Contents** in the Project Navigator.)
(Synopsys) *VHDL Reference Guide*	(PDF file only) This guide describes doing VHDL designs and synthesizing them with Synopsis' FPGA Express. (Can be accessed from **Help** → **Foundation ISE Help Contents** in the Project Navigator.)
ABEL Reference Guide	(PDF file only) This guide contains general information on ABEL as well as CPLD-specific information on ABEL. (Can be accessed from **Help** → **Foundation ISE Help Contents** in the Project Navigator.)

Table 1-1 Foundation Series ISE 3.1i Online Book Collection

Title	Description
Implementation-Related Books:	
Constraints Editor Guide	This manual describes the Xilinx Constraints Editor GUI that can be used *after* the design has been implemented to modify or delete existing constraints or add new constraints to a design.
Development System Reference Guide	This book describes the Xilinx design implementation software. This includes detailed informaiton on the programs that generate EDIF files, LCA files, and BIT files. The book covers all the command line program options and files that are generated by these programs. This is the processing information you see in the Transcript window as a Foundation Series ISE design is being implemented. It also contains in-depth information on timing constraints, including examples.
FPGA Editor Guide	The FPGA Editor is a graphical editor used to display and configure FPGAs. The FPGA Editor enables you to place and route critical components before running automatic place and route tools on an entire design, modify placement and routing manually, interact with the physical constraints file (PCF) to create and modify constraints, and verify timing against constraints.
Floorplanner Guide	This book describes the Floorplanner, a graphical interface tool to help you improve performance and density of your design.
Hardware User Guide	This manual describes the Xilinx Demonstration hardware and its associated software interfaces. The hardware includes the FPGA and CPLD demonstration boards, which are used for design verification.
Timing Analyzer Guide	This manual describes Xilinx's Timing Analyzer program, a graphical user interface tool that performs static analysis of a mapped FPGA or CPLD design. The mapped design can be partially or completely placed, routed, or both.

Table 1-1 Foundation Series ISE 3.1i Online Book Collection

Title	Description
Device Programming Books:	
JTAG Programmer Guide	This guide documents the graphical interface used for in-system programming and verification of CPLD and FPGA parts. The guide also describes how to set up and use JTAG download cables.
Hardware Debugger Guide	(FPGAs only) This manual describes how to program, verify, and debug FPGA devices. It describes the XChecker, MultiLINX, and Parallel III cables and explains how to connect the cable pins to your target device for various functions: downloading, verification, and debugging. It also includes a tutorial for debugging a design using the demonstration boards as target devices.
PROM File Formatter Guide	(FPGAs only) This manual explains how to use a Windows-based tool to format bitstream files into HEX format files compatible with Xilinx and third-party PROM programmers. You use the PROM files to program a PROM device, which is then used to configure daisy chains of one or more FPGAs for one application (configuration) or several applications (reconfiguration).

Chapter 2

Design Environment

This chapter provides an overview of the Foundation Series ISE 3.1i concepts, tools, and features that comprise the Foundation Series ISE 3.1i design environment. It contains the following sections:

- "Projects"
- "Project Management"
- "Design Flow"

Projects

With Foundation Series ISE, your FPGA or CPLD design is organized and tracked as a *project*. When you create a new project, you must specify the following:

- Project directory

  Each project should have a unique directory to store its source files, intermediate data files, and resulting files.

- Targeted Xilinx device family (architecture)

  Virtex, Spartan, XC9500 are examples of device families.

- Targeted Xilinx device in the specified device family

  An example device is "S05 PC844-4" where S05 designates a specific Spartan device, PC844 specifies the package, and -4 indicates the speed grade.

- Synthesis tool

  Every project includes synthesis. FPGA Express and XST (Xilinx Synthesis Technology) are the synthesis tools. VHDL / Verilog / ABEL versions are available for these tools.

The preceding items become associated with the project as *project properties*. They enable the *Project Navigator*, the main user interface for Foundation Series ISE (see Figure 4-1 in the "Project Navigator" chapter), to display and run only those processes appropriate for the targeted device and synthesis flow.

Your Foundation Series ISE project contains all the logical descriptions (HDL files, state diagrams, and / or schematics) for programming the specified Xilinx device as well as any documentation files, simulation models, and test files. All these are the *sources* for the project. You create and add the sources to your project.

To perform an action on a source in your project, you select the source and then select the *process* you want to perform on that source. The *processes* you can perform on a particular source to test and implement your design are kept track of for you and run by the Foundation Series ISE Project Navigator.

You can *archive* or take a *snapshot* of a project at anytime to preserve that version / revision of the project.

Project Management

The Project Navigator is the primary interface for Foundation Series ISE. It provides project management, tool integration, and flow control. From the Project Navigator, you identify all the sources for the project. Sources include schematics and HDL design files, as well as specification documents and test files. You also identify the target device and select a project flow.

Based on the selected target device and project flow, the Project Navigator organizes all the parts of your design. It keeps track of the processes necessary to move the design from the conceptual stage through to implementation in the targeted Xilinx device. You can launch text and schematic editors, navigate through the hierarchy of the design, simulate your design, place-and-route your design, and much more from the Project Navigator.

Project Navigator's auto-make feature monitors dependencies between process steps and automatically runs and updates the intermediate processes when necessary.

The following figure shows an example of the Project Navigator windows for a project (Project.npl).

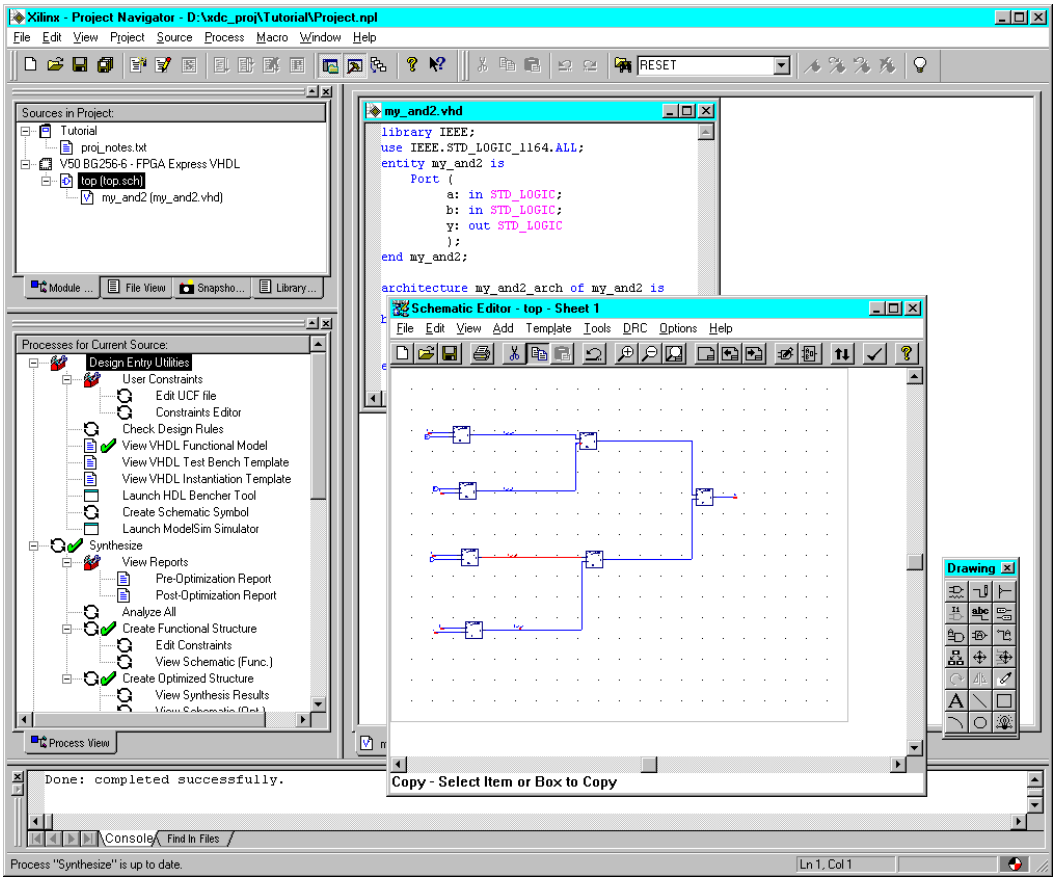


Figure 2-1 Foundation Series ISE Project Navigator - Virtex Design with XST VHDL Synthesis

The project in Figure 2-1 is titled “Tutorial” in the Source window. The *Source window* contains all the design files associated with your project; icons indicate the source type. In Figure 2-1, the information in the Source window indicates that this design, described as “Tutorial,” is for a Virtex device (V50 BG 256-6) and is using the XST VHDL tool. It has a top level schematic `top.sch` (its source is shown in the *Schematic Editor window*) and an underlying VHDL module `my_and2.vhd` (shown in the *HDL Editor* window behind the Schematic Editor window).

The *Process window* (below the Source window) displays the available processes for the selected source. Icons indicate what kind of process it is. The current state of each process is also indicated. A green check mark indicates a successfully completed, up-to-date process. A red X indicates an error. An exclamation point indicates a warning. In Figure 2-1, a green check mark precedes the processes that have been successfully performed on the top.sch module.

There is a *Transcript window* at the bottom of the Project Navigator that logs project processing and reports errors. Red icons indicate navigable errors. Yellow icons indicate navigable warnings. You can navigate to the source of the error / warning or to solution records on the Xilinx support website pertaining to the problem.

One of the most powerful features of the Project Navigator is that it is context sensitive in regards to source file types, device types, and synthesis tool selection. The steps in the Process window reflect this context sensitivity. For example, if you highlight a text file describing the project, as shown in Figure 2-2, there is no processing to perform. Therefore, the Process window contains no processes. Contrast that Process window to the one in Figure 2-1 in which the top level module of the design is highlighted and all the processes that can be performed from that point in the design hierarchy are listed.

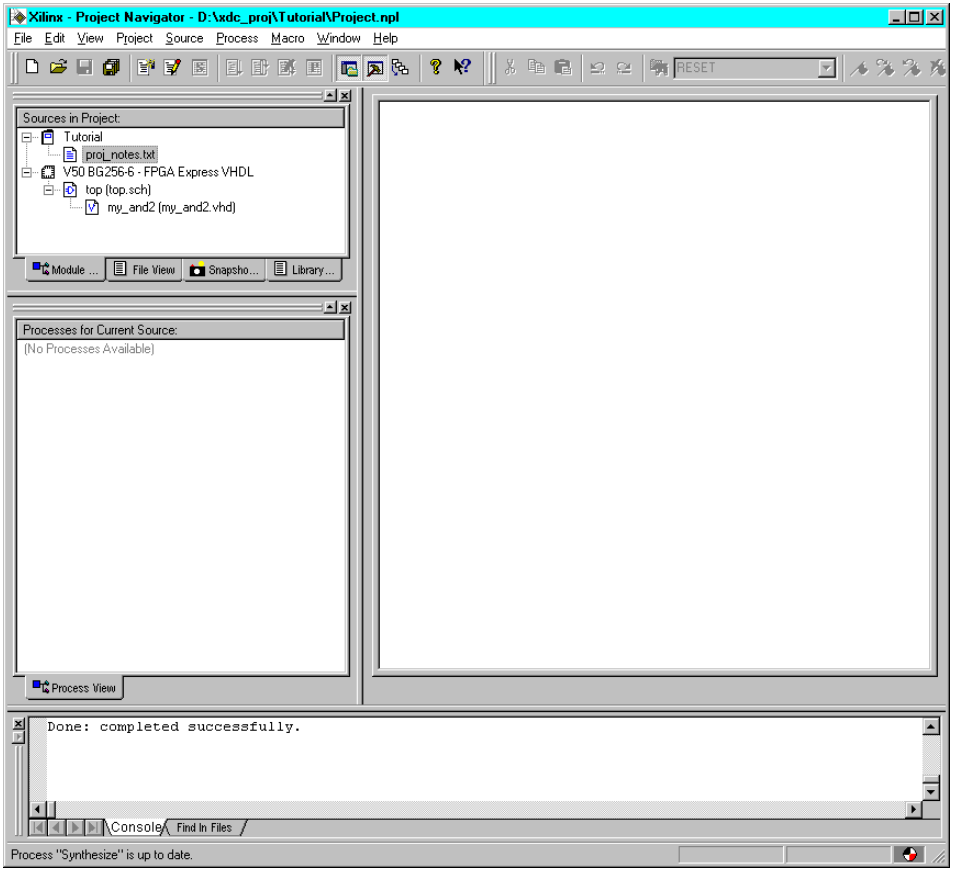


Figure 2-2 Processes for a Project Documentation Text File

Design Flow

The design flow is a multi-step, iterative process that includes the following stages: design entry, synthesis, implementation, programming file creation, and bitstream downloading. Two simulation points are available to test your design. The first is a functional simulation to check the logic prior to synthesis. The second is post-route simulation to verify that the design meets the timing requirements you set for your design in the targeted device. By using the same testbench and comparing the results at each of the simulation stages, you

can be confident that what was synthesized and optimized is what was desired for your design.

Design Entry

You can create your design using schematics, text-based entry (HDL code), and/or state diagrams. You can create new sources or add *remote sources* (existing sources from other projects) for your project. Behavioral simulators are available to test the logic of your designs before continuing to the next stages.

Design Entry Tools

Foundation Series ISE includes the following design entry tools.

HDL Editor

The HDL Editor is integrated into the Project Navigator. The HDL Editor is a language sensitive text editor for VHDL, Verilog, and ABEL- HDL. (ABEL-HDL is supported for CPLDs only). The editor includes color coding and context sensitive help for reserved words. The editor's "New source" wizard can build the initial text structure for your HDL file. The Project Navigator also includes a Language Template feature with pre-built language and synthesis templates to assist with HDL entry.

ECS Schematic Editor and Symbol Editor

For schematic designs, the 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 is included within the Schematic Editor for creating or customizing a variety of electrical symbol types.

StateCAD (State Diagrams)

You can use StateCAD® and StateBench® from Visual Software Solutions, Inc. (VSS) with the Project Navigator if you want to create state diagrams for state machine designs. The Xilinx Limited Edition of StateCAD/StateBench is available on the Star Partner CD included in the Foundation Series ISE package. Or, you can point the Project Navigator to your existing VSS tools. The Project Navigator launches StateCAD for source creation or modification. You can add State

diagrams (as user documents) and / or their corresponding StateCAD generated HDL source modules to your project. Once added, they are updated automatically in the Project Navigator whenever they are modified within StateCAD. You can use the StateBench simulator to verify the behavior of your state diagram.

LogiBLOX

Foundation Series ISE includes access to LogiBLOX to aid in the creation of high-level modules, such as shift registers, and multipliers. LogiBLOX supports XC4000, Spartan, SpartanXL, and CPLD devices only.

CORE Generator System

Access to the Xilinx CORE Generator System is available through Foundation Series ISE. The Xilinx CORE Generator System is a graphical interactive tool that generates and delivers parameterizable cores optimized for the following Xilinx FPGA device families: XC4000, Spartan, Spartan2, Virtex, and Virtex2.

Functional Simulation

ModelSIM simulators are supported in the Project Navigator for functional simulation with or without a testbench / test fixture. A testbench / test fixture template generating tool is available in the Project Navigator. For automated testbench / test fixture creation, you can use the HDL Bencher from VSS.

HDL Bencher (Test Bench/Test Fixture Creation)

HDL Bencher is an automated testbench / test fixture creation tool from VSS. The Xilinx Edition of the HDL Bencher is included on the Star Partner CD included in your Foundation Series ISE package. Or, you can point the Project Navigator to an existing HDL Bencher installation. All editions of the HDL Bencher are fully integrated with the Project Navigator.

ModelSIM (Functional Simulation)

ModelSIM from Model Technology, Inc. is integrated in the 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 Foundation Series ISE package.

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 incorporated with Foundation Series ISE: XST and FPGA Express. You select the synthesis tool when you create a project. The synthesis flow for your project is different for each tool.

Synthesis Tools

Foundation Series ISE includes two synthesis tools: XST and FPGA Express

XST (Xilinx Synthesis Technology)

XST is a Xilinx tool that synthesizes HDL designs to create EDIF netlists. The Project Navigator invokes XST processing when you select a source and then select a synthesis process for a project that has an 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

FPGA Express from Synopsys 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 an FPGA Express synthesis tool associated with it. Depending on the Foundation Series ISE product you purchased, the Express Constraints Editor (pre-optimization), Time Tracker (post-optimization), and Library Viewer 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.

Note 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.

Implementation

The implementation stage consists of converting the logical design file format, EDIF, created in the design entry stage into a physical file format for a specific Xilinx architecture. Foundation Series ISE includes the Xilinx implementation tools to perform the necessary translate to bitstream generation functions for your design.

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

You can invoke the Xilinx Constraints Editor to add timing and location constraints for the implementation of your design. You can also invoke 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.

Post-Route Simulation

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

Programming File Creation

When your design meets all your requirements, you can create a programming file that can be downloaded to the project's target device. The Xilinx Hardware Debugger, JTAG Programmer, and PROM File Formatter can be invoked from the Project Navigator.

Chapter 3

Creating a Project

To start a design, you must first create a project that represents your design. The project creation process includes specifying a directory for the project, identifying the Xilinx device you want to target for your design, choosing a project flow, and adding/creating source files to the project.

This chapter contains the following sections that describe the project creation process:

- "Specifying a Project Name and Location"
- "Selecting a Device and Synthesis Tool"
- "Changing the Targeted Device"
- "Changing the Synthesis Tool"
- "Creating/Adding Source Files"
- "Source Types"

Specifying a Project Name and Location

The first step in creating a new project is to specify a project name and to identify a directory to hold all of the project's files.

Note Each project must have a separate, unique directory containing only one project file (*project_name*.npl).

1. Open the Project Navigator.

   To open the Project Navigator, select **`Start`** → **`Programs`** → **`Foundation Series ISE 3.1i`** → **`Project Navigator`**. Or, click on the Foundation Series ISE 3.1i icon on your PC desktop.

2. Select **File** → **New Project** from the Project Navigator menu bar.
3. The New Project dialog box appears. An example is shown in the following figure.

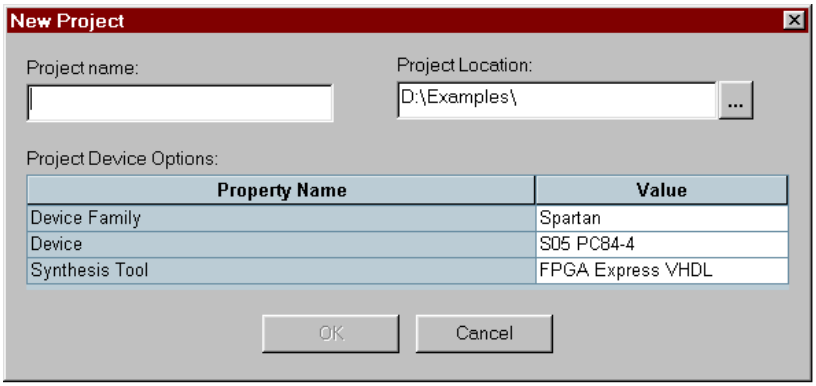


Figure 3-1 New Project Dialog Box

4. 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).
5. As you type in the Project Name field, what you type is also entered in the Project Location field. It is added at the end of the currently displayed path name to create a subdirectory for the project.

   For example, if the Project Location box currently has **D:\Examples** and you type **NewProject** as the Project Name, the Project Location then becomes **D:\Examples\NewProject**. In this case, the project file **NewProject.npl** is created and placed in **D:\Examples\NewProject**.

   To select a different location for the project such as **D:\Project1** instead of **D:\Examples\NewProject**, you can modify the path in the Project Location field directly. Or, if the directory already exists, you can click in the Project Location field and use the browse button that appears to select the directory. Then, if **NewProject** is in the Project Name field, the project file **NewProject.npl** is created and placed in the **D:\Project1** directory.

If you modify anything in the Project Name field *after* selecting the Project Location directory, the path in the Project Location field is updated to reflect the changes you made to the Project Name. For example, if **D:\Project1** is in the Project Location field and then you type **MyProject** in the Project Name field, the Project Location field becomes **D:\Project1\MyProject**.

Note If the directory shown in the Project Location field does not currently exist, it is created. If the directory exists and it already contains a project file with the same name as the one you entered in the Project Name field, you are prompted as to whether you want to overwrite the existing project file.

6. After you enter the desired project name and location, you can click **OK** to create the project. The project will target the default device and use the default synthesis tool identified in the Project Device and Synthesis Tool selection area of the New Project dialog box.

   You can click in the Value fields for the Device Family, Device, and Synthesis Tool properties and select different values from the pull-down lists that appear before clicking **OK** to create the project. For example, if you want to target a device in the Virtex device family, select **Virtex** from the Device Family pull-down menu. Then select the specific Virtex device you want from the Device pull-down menu. For example, if you want to use the Virtex V50 device, BG256 package, and -6 speed grade, select **V50 BG256-6**. Finally, select the synthesis tool you want to use from the Synthesis tool pull-down menu, for example, **XST VHDL**.

   Refer to the “Selecting a Device and Synthesis Tool” section for information on the inter-relationship of the device and synthesis tool selection for your project.

Figure 3-2 shows the Project Navigator’s windows for a newly created project.

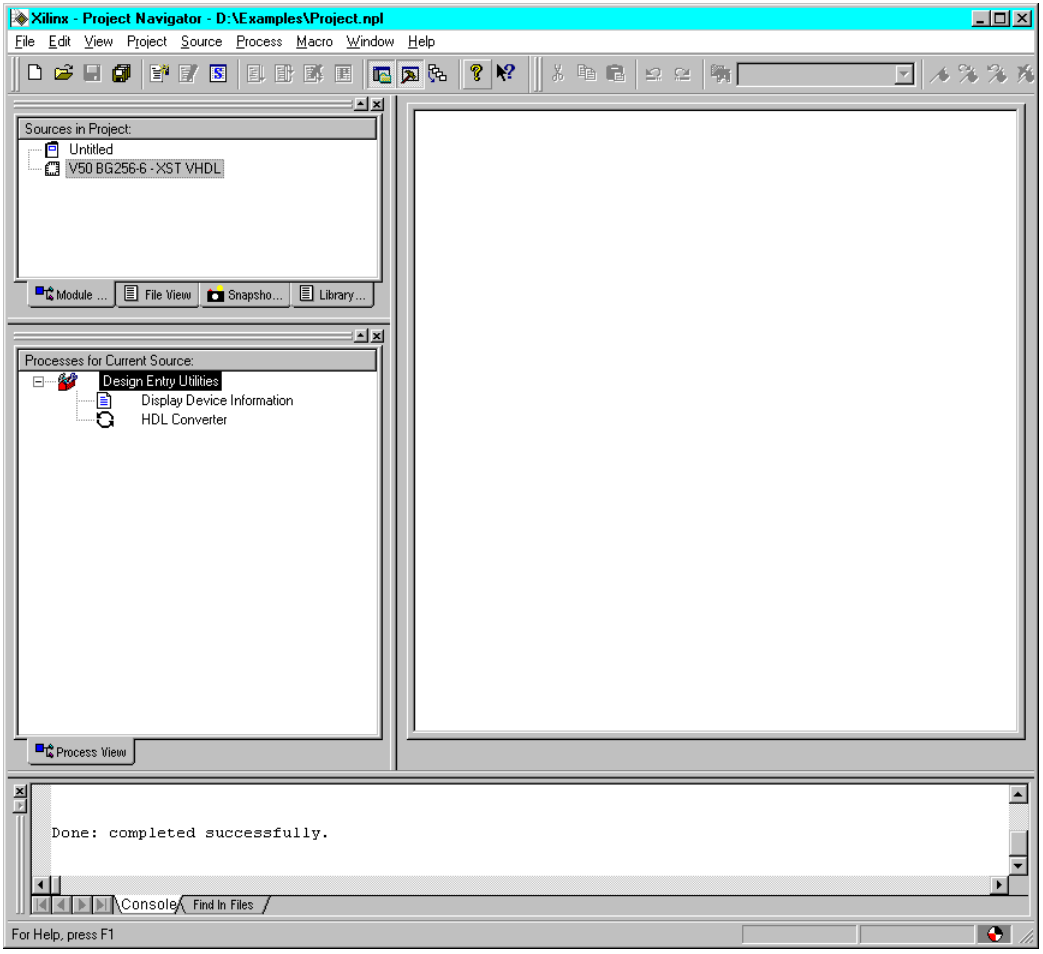


Figure 3-2 Project Navigator Windows for a Newly Created Project

Selecting a Device and Synthesis Tool

Whenever you create a new project, a default device and synthesis tool is assumed for the project. The default device and synthesis tool appear in the Device Family, Device, and Synthesis tool fields of the New Project dialog box (see the "Specifying a Project Name and Location" section). The first time you create a new project a default device and synthesis tool are shown, for example, the Spartan S05 PC84-4 device and FPGA Express VHDL. After that, the device and synthesis tool you used for your last project are used as the default for the new project.

You can change the device and synthesis tool in the New Project dialog box when you create the project. Or, you can change them later as described in the "Changing the Targeted Device" section and the "Changing the Synthesis Tool" section.

Table 3-1 shows the available synthesis tools for each device. The sections after the table briefly describe the project flow characteristics for each synthesis tool.

Table 3-1 Available Synthesis Tools for Each Device Family

Device Family	Synthesis Tool					
	XST VHDL	XST Verilog	FPGA Express VHDL	FPGA Express Verilog	ABEL XST	ABEL BLIF
FPGAs:						
Spartan			X	X		
Spartan2	X	X	X	X		
SpartanXL			X	X		
Virtex	X	X	X	X		
Virtex2	X	X	X	X		
VirtexE	X	X	X	X		
XC4000E			X	X		
XC4000EX			X	X		
XC4000L			X	X		
XC4000XL			X	X		
XC4000XLA			X	X		

Table 3-1 Available Synthesis Tools for Each Device Family

Device Family	Synthesis Tool					
	XST VHDL	XST Verilog	FPGA Express VHDL	FPGA Express Verilog	ABEL XST	ABEL BLIF
CPLDs:						
XC9500	X	X	X	X	X	X
XC9500XL	X	X	X	X	X	X
XC9500XV	X	X	X	X	X	X

XST VHDL

The XST VHDL project flow has the following characteristics:

- 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
- Supports Virtex, Virtex2, VirtexE, Spartan2, and XC9500/XL/XV devices

XST Verilog

The XST Verilog project flow has the following characteristics:

- 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
- Supports Virtex, Virtex2, VirtexE, Spartan2, and XC9500/XL/XV devices only.

FPGA Express VHDL

The FPGA Express VHDL project flow has the following characteristics:

- 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
- Supports Virtex, Virtex2, VirtexE, Spartan2, Spartan, SpartanXL, XC4000E/L/EX/XL/XLA, and XC9500/XL/XV devices
- Certain configurations include the Express Constraints Editor, Express Time Tracker, and Schematic Viewer tools from Synopsys

FPGA Express Verilog

The FPGA Express Verilog project flow has the following characteristics:

- 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
- Supports Virtex, Virtex2, VirtexE, Spartan2, Spartan, SpartanXL, XC4000E/L/EX/XL/XLA, and XC9500/XL/XV devices
- Certain configurations include the Express Constraints Editor, Express Time Tracker, and Schematic Viewer tools from Synopsys

ABEL XST

The ABEL XST project flow has the following characteristics:

- Uses Xilinx Synthesis Technology (XST) for synthesis
- Can contain only ABEL-HDL code. No VHDL or 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
- Is supported for CPLD (XC9500/XL/XV) designs only.

ABEL BLIF

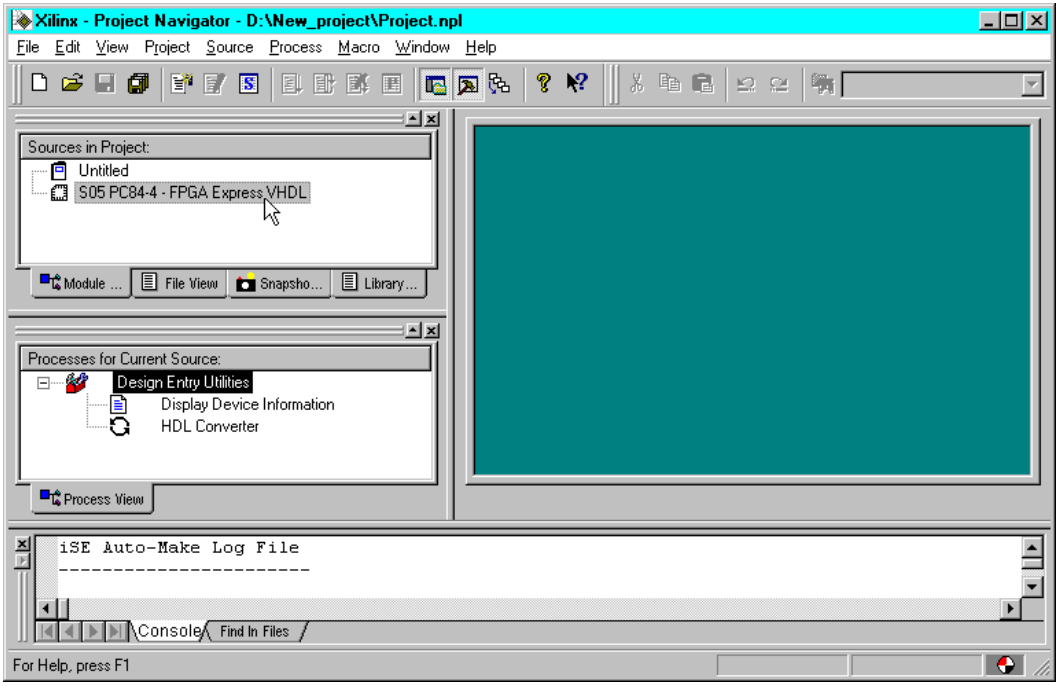
The ABEL BLIF project flow has the following characteristics:

- Uses the ABEL-HDL compiler engine for synthesis
- Can contain only ABEL-HDL code. No VHDL or Verilog is allowed.
- Creates a BLIF-based netlist for ABEL-HDL code prior to synthesis
- Creates a functional VHDL model for schematics prior to synthesis
- Is supported for CPLD (XC9500/XL/XV) designs only.

Changing the Targeted Device

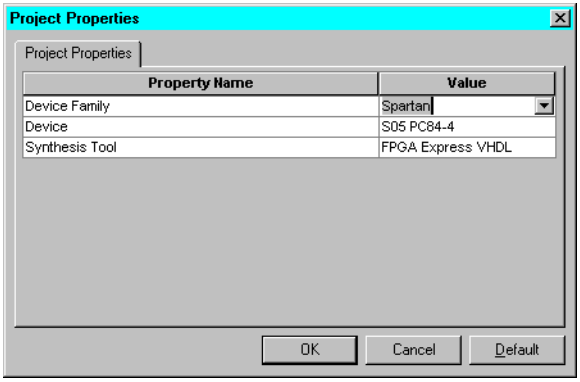
You can change the device you want your design to target at anytime. Use the following procedure to select a new Xilinx device for an existing project.

1. Click on the Device / Synthesis Tool line in the Sources Project window to highlight that source.

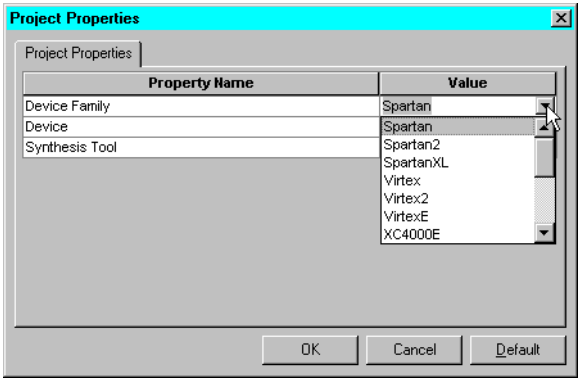


2. Select **`Source`** → **`Properties`** from the Project Navigator menu. (You can also right-click on the Device / Synthesis Tool line and then select **`Properties`** from the box that appears.)

3. A Project Properties dialog box appears with the current values identified in the Value fields.



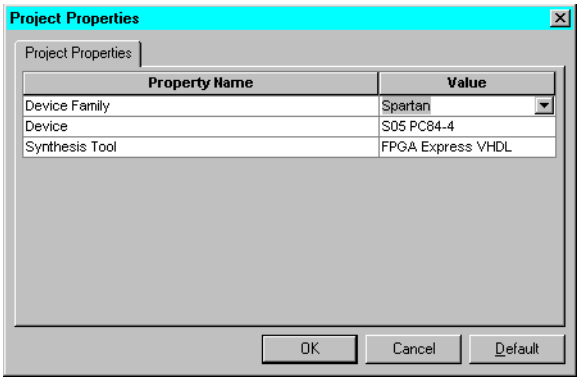
4. Click on the scroll button on the right side of the Device Family Value field to display the device family list.



5. Scroll through the device family list and select the desired device by clicking on its name.

6. As soon as you select a device family, the Project Properties dialog box automatically updates with a default device and appropriate synthesis tool 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.

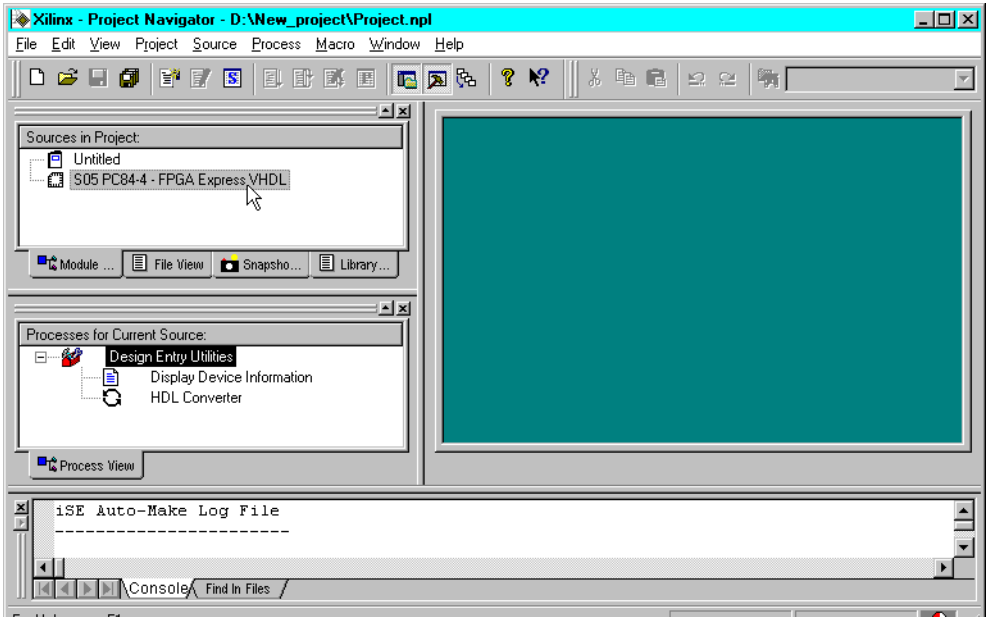


7. Modify the defaults as desired by placing the cursor on the right side of a Value field and scrolling through the list that appears.

Changing the Synthesis Tool

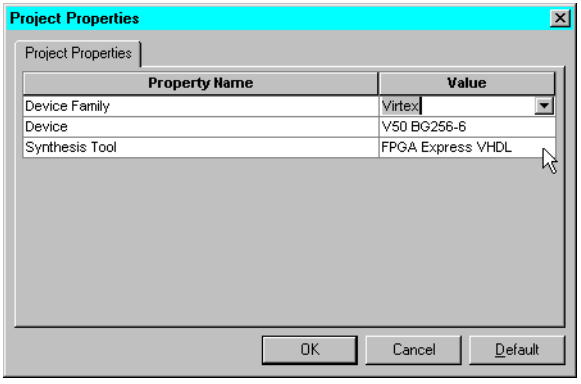
The selection of a synthesis tool is closely linked to the Xilinx device family and the type of design (VHDL or Verilog) you want for your project. Be aware that changing the synthesis tool may require changing the targeted device. You can change the synthesis tool at anytime using the following procedure:

1. Click on the Device / Synthesis Tool line in the Source window to highlight that source (indicated by the arrow in the following figure).

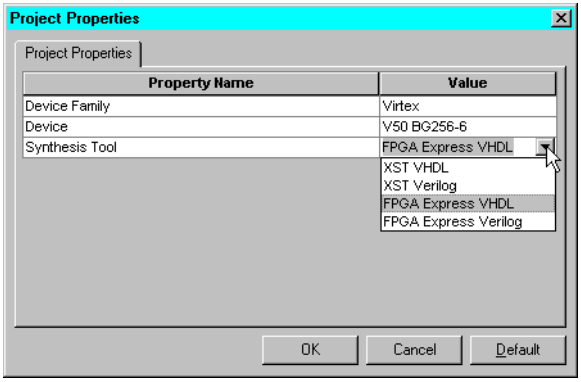


2. Select **`Source`** → **`Properties`** from the Project Navigator menu. (You can also right-click on the Device / Synthesis Tool line and then select **`Properties`** from the box that appears.)

3. A Project Properties dialog box appears with the current values shown in the Value fields.



4. Click on the right side of the Synthesis Tool Value field to display a list of the synthesis tools supported for the selected Device Family.



5. Scroll through the synthesis tool list and select the desired tool by clicking on its name. Only synthesis tools appropriate for the selected device family are listed

Creating/Adding Source Files

After you create your project and select a Xilinx device and synthesis tool, you can begin creating and / or adding source files for your project. A *source* is any element that contains information about a design. Sources include the files necessary to describe the behavior of your design (schematics or HDL files), files needed to test your design (testbenches or test fixtures), and other design documentation. The types of sources (schematic, VHDL, Verilog, ABEL) available in a project vary depending on the selected device and synthesis tool. Refer to the “Source Types” section for information on the types of sources supported for Foundation Series ISE projects.

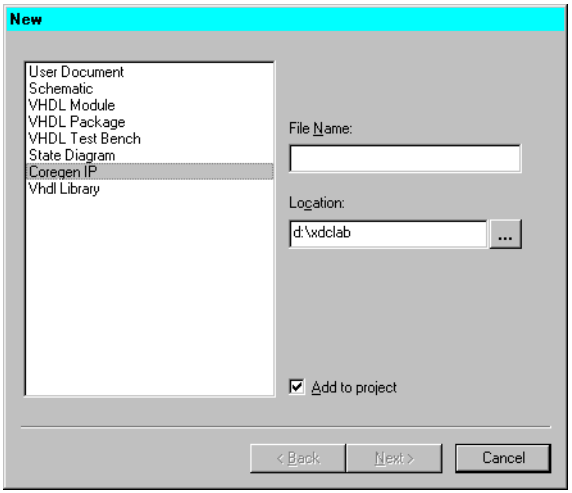
You can add existing files or create new files for your project using the procedures described in the following sections.

Creating a New Source

To create a new source file for a project, do the following:

1. Open a project and click on any line in the Sources window to highlight it.
2. Select **`Project`** → **`New Source`** from the Project Navigator’s Menu. (Or, you can right click on any source in the Sources window and select **`New Source`** from the pull-down menu that appears.)

3. The New dialog box appears with a list of appropriate sources for the project's target device and synthesis tool. The following figure shows the available sources for a Virtex XST VHDL project.



4. Select the type of source you want to create from the list by clicking on it.

5. Enter a name for the new source file in the File Name field. Do not add an extension to the file name. Foundation Series ISE adds the appropriate extension for the selected source type.

   Sources cannot have spaces or periods in their names.

   Note State diagram names can only be eight characters and must start with an alphabetic character.

6. Check the "Add to Project" box to add this source to the project automatically after it is created.

   Note The "Add to Project" box does not apply to all source types. You must add certain sources, such as state diagrams and test-benches / text fixtures, to the project manually as described in the "Adding an Existing Source to the Project" section.

7. Click **`Next`** to continue.

8. The "New Source Information" window usually appears at this point to summarize the requested new source.

If you are adding a VHDL Module or Verilog Module, the Define VHDL Source or Define Verilog source wizard appears before the "New Source Information" window. Refer to "Creating HDL Source Files" section of the "HDL Sources" chapter for information on using the HDL source wizard. Click **Next** in the Define Source window to continue.

Click **Finish** in the New Source Information window to continue.

9. The type of source you select determines what happens next. For example, if you are creating a schematic, the Schematic Editor opens. If you are creating a User Document, a Notepad window (or your usual text editor) opens for text entry.

   A source file named as specified in the File Name field is loaded in the selected source-creation application ready for you to create the source. If the "Add to Project" box was checked, the source is automatically added to the project and is listed appropriately in the source window.

10. Create the source.

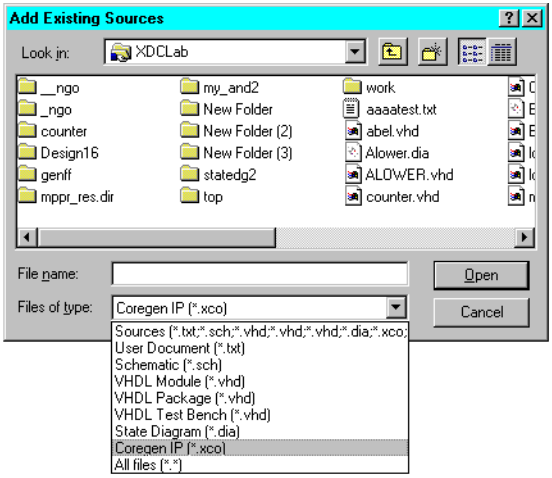
Adding an Existing Source to the Project

Use the following procedure to add an existing source to a project. The source can be in the project directory or in a remote directory (a directory other than the project directory). The source is not moved or copied from its current directory.

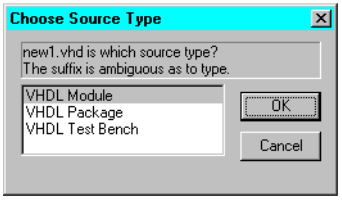
1. Open a project and click on any line in the Sources window to highlight it.
2. Select **Project** → **Add Source** from the Project Navigator's Menu. (Or, you can right click on any source in the Sources window and select **Add Source** from the pull-down menu that appears.)
3. Use the Add Existing Sources window to browse to the source you want to add to the project.

   You can click on the right side of the "Files of type" box and select a source type from the drop-down list that appears. This limits the display to the specified type of source.

Note The source you select here remains in its current directory. It is not moved or copied to the project's directory if it is not already there.



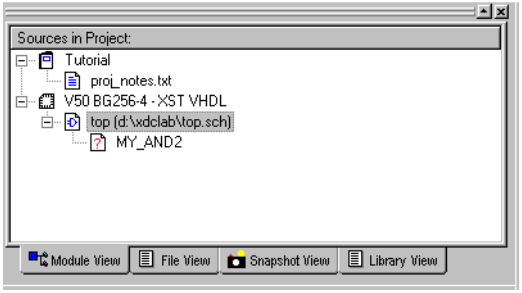
4. After selecting the file, the Choose Source Type dialog box may appear. If it does, you need to identify the file type you are adding. For example, a file with the .vhd extension could be a VHDL Module, testbench, or package.



5. As soon as you click **Open** to select a file in the Add Existing Sources window or **OK** in the Choose Source type window, the file appears in the Sources window for the current project.

The directory path appears along with the filename for all remote sources (sources not in the current project directory). You can truncate or expand the path information that appears by selecting **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.

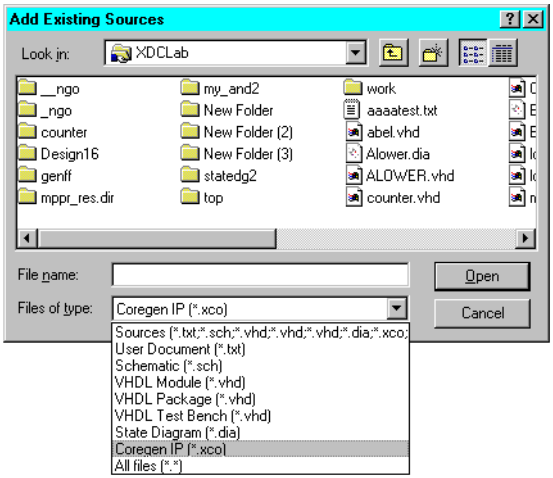


Adding a Copy of an Existing Source to the Project

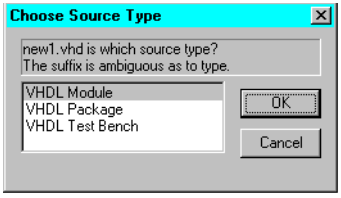
Use the following procedure to add a copy of an existing source to a project. The source can be in the project directory or in a remote directory (a directory other than the project directory). If the source is in the project directory, a second copy is not made. The original is added to the project. 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.

1. Open a project and click on any line in the Sources window to highlight it.
2. Select **`Project`** → **`Add Copy of Source`** from the Project Navigator menu. (Or, you can right click on any source in the Sources window and select **`Add Copy of Source`** from the pull-down menu that appears.)
3. Use the Add Existing Sources window to browse to the source file you want to add to the project.

   You can click on the right side of the “File of type” box and select a source type from the drop-down list that appears. This limits the display to the desired type of source.

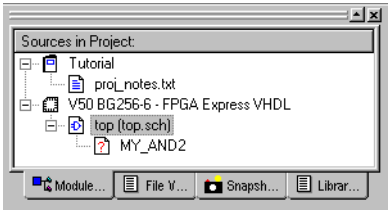


4. After selecting the file, the Choose Source Type dialog box may appear. If it does, you need to identify the file type you are adding. For example, a file with the .vhd extension could be a VHDL Module, testbench, or package.



5. As soon as you click **Open** to select a file in the Add Existing Sources window or **OK** in the Choose Source type window, a copy of the file is created and placed in the project directory (if the file is not already in the project directory). The file is added to the project and appears in the Sources window.

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



Source Types

As described in this chapter, a source is any element that contains information about a design. Sources include the files necessary to describe the behavior of your design (schematics or HDL source files), files needed to test your design (test benches, test fixtures, waveforms for simulation), and general project design documentation. Sources are listed in the Source window. Refer to the "Source Window" section of the "Project Navigator" chapter for information on the Source window. Icons to the left of the source filename identify the type of design source. Refer to the online Help for a list of icons that identify each design source type.

Your projects could consist of the types of sources listed in the following table.

Table 3-2 Project Navigator Source Types and File Extensions

Source Type	File Extension
Project file	.npl
User document (such as a specification)	.txt, .wri, .doc, .xls, .hlp (or any extension not recognized by the Project Navigator)
Schematic	.sch
State diagram (currently a "user document")	.dia
VHDL module (includes VHDL logic description for state diagrams, schematics, etc.)	.vhd
VHDL package	.vhd

Table 3-2 Project Navigator Source Types and File Extensions

Source Type	File Extension
VHDL testbench	.vhd
Waveform stimulus	.wdl
Verilog module (includes Verilog logic description for state diagrams, schematics, etc.)	.v
Verilog test fixture	.tf
ABEL-HDL logic description	.abl
ABEL-HDL test vectors	.abv (or .abl)
Coregen IP	.xco
LogiBLOX module	.mod

Source Type Descriptions

The various source types you can add to your project are described in the following sections. All sources appear in the Project Navigator's Source window (see Figure 3-3 for an example Source window). Refer to the "Source Window" section of the "Project Navigator" chapter for detailed information on how to access and display source data.

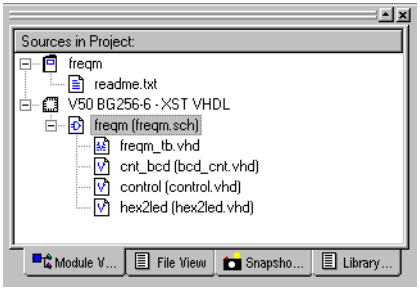
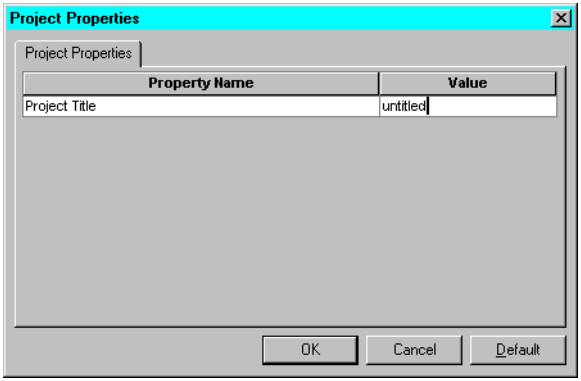


Figure 3-3 Example Source Window for a Virtex XST VHDL Project

Project Title

The project title is in the first line of the Source window ("freqm" in Figure 3-3). By default, the project title is "untitled."

To change the project title, click on the first line in the Source window and then select **Source** → **Properties** to access the Properties dialog box shown in the following figure.



User Documents

User documents are listed beneath the project title. *User document* is a catch-all source type used for any file that you want to have associated with a design project but that should not be or cannot be processed by Foundation Series ISE. The most common use of this source type is for text documents (.txt, .wri, .doc, etc.) that describe the project. Currently, state diagrams (.dia) are also "user documents" because they are not processed by the Project Navigator.

You can select **Project** → **New Source** → **User Document** to use the Project Navigator's New source wizard to create a new, blank text file and open it in your usual Window 98 / NT / 2000 text editing tool. Refer to the "Creating / Adding Source Files" section for detailed information on creating new sources and adding existing sources to your project. Whenever you create or add a file and designate it as a *user document*, it is listed under the project title when it is added to the project. It is not processed by the Project Navigator.

Device/Synthesis Tool

The device / synthesis tool line is identified in the Source window by the "chip" icon that appears in front of it. The relationship between the device and synthesis tool is discussed in the "Selecting a Device and Synthesis Tool" section. The type of sources available for your project vary depending on the device you are targeting for the design

and on the synthesis tool you selected for the target device as shown in the following table.

Table 3-3 Source Type vs. Synthesis Tool

Source Type	Synthesis Tool					
	XST VHDL	XST Verilog	FPGA Express VHDL	FPGA Express Verilog	ABEL HDL	ABEL BLIF
Project Title[1]	X	X	X	X	X	X
User document	X	X	X	X	X	X
Device / Synthesis Tool[1]	X	X	X	X	X	X
Schematic	X	X	X	X	X	X
VHDL Module	X		X	X		
VHDL Test Bench	X		X	X		
VHDL Package	X		X	X		
VHDL Library	X		X	X	X	X
Verilog Module		X	X	X		
Verilog Test Fixture		X	X	X		
ABLE HDL Module (CPLDs only)					X	X
ABEL Test Vector (CPLDs only)					X	X
State Diagram	X	X	X	X	X	X
Coregen IP (FPGAs only)	X	X	X	X		
LogiBlox Module (XC4000s, Spartan, SpartanXL, and CPLDs only)	X	X	X	X	X	X

[1]These are required “sources” in all projects and are added automatically. Their values are set as Project Properties. To modify either, right-click on its line in the Source window and select **Source** → **Properties**.

To change the device and / or synthesis tool, click the Device / Synthesis Tool line and then select **`Source`** → **`Properties`** from the Project Navigator menu. When the Properties dialog box appears, use the pull-down menus in the Values field to select the desired device family, device, and / or synthesis tool. Refer to the "Changing the Targeted Device" section and the "Changing the Synthesis Tool" section for illustrated instructions.

The Project Navigator associates the device / synthesis tool and all sources listed under it with *processes* it can perform on them. To view the available processes for a source, click on the Device / Synthesis Tool line or any source listed under it and then check the Process window for the list of available processes.

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 to the project as a user document. You can add the optimized HDL module (.vhd or .v) translated from the state machine to the project as a VHDL or Verilog source file. After the state diagram file and / or its corresponding HDL file are added, they are updated whenever modifications are made to either of them using the state diagram tool.

Foundation Series ISE includes support for StateCAD from Visual Software Solutions, Inc. for the creation and development of state machines and their translation to HDL code.

Schematic

Schematics (.sch) are another form of design entry. Schematics are created in the ECS Schematic Editor 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 shown in the Source window. You can view the functional module by clicking on a schematic source and then clicking View VHDL (or Verilog) Functional Model in the Process window.

VHDL Module

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

VHDL Test bench

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

A VHDL test bench 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.

Refer to the "Creating a Testbench / Test Fixture" section of the "Simulation" chapter for information on creating VHDL test benches for your project.

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 precompiled and can be included in VHDL designs. These packages are intended for use during synthesis, but may also be used for simulation. Refer to the *XST User Guide* for a list and description of supported predefined packages.

VHDL Library

VHDL requires all design sources to be in a library. Refer to the "HDL Library Mapping" chapter for information on creating and naming VHDL libraries for use in your project.

Verilog Module

A Verilog module (.v) is a file that contains code for a single Verilog module. Refer to the “HDL Sources” chapter for information on creating Verilog source files for your project.

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 Foundation Series ISE, Verilog test fixture sources are associated with the source file that they instantiate.

Refer to the “Creating a Testbench / Test Fixture” section of the “Simulation” chapter for information on creating Verilog test fixtures for your project.

ABEL-HDL Module (CPLDs Only)

An ABEL-HDL module (.abl) is a file containing ABEL code. Refer to the “HDL Sources” chapter for information on creating HDL source files for your project.

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 parameterizable 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.

Refer to the “CORE Generator” chapter for information on using COREs in your project.

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 regis-

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

Refer to the “LogiBLOX” chapter for information on using LogiBLOXs in your project.

Chapter 4

Project Navigator

The Project Navigator is the primary user interface for Foundation Series ISE. It integrates the design entry, implementation, synthesis, and simulation tools and processes to facilitate design production.

This chapter contains the following sections that describe the Project Navigator functions and windows:

- “Starting the Project Navigator”
- “Project Navigator Windows”
- “Source Window”
- “Process Window”
- “HDL Editor Workspace”
- “Error Navigation from the Transcript Window”
- “Integrated Tools”
- “Customizing the Project Navigator”
- “Docking / Undocking Project Navigator Windows”

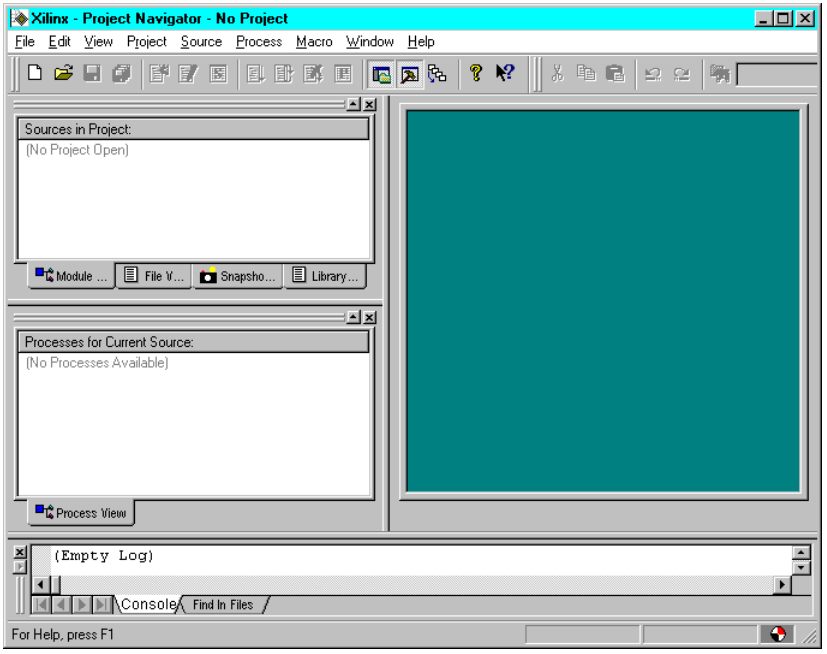
Starting the Project Navigator

You can use any of the following methods to open the Project Navigator:

- Click on the Project Navigator icon (shown below) on your desktop.

- Select **`Start`** → **`Programs`** →**`Foundation Series ISE 3.1i`** → **`Project Navigator`** from your PC desktop.
- For existing projects, you can start from Windows Explorer. Go to the project directory and double-click on the project's .npl file. This opens the Project Navigator and loads the project.

The very first time you open the Project Navigator all of its windows are blank. After that, the Project Navigator opens with the last project you worked on.



To begin using the Project Navigator, you must first create a project and then create and / or add source files for the project. To create a project, select **File** → **New Project** from the Project Navigator menu. The steps necessary to create a new project and to add sources are described in the “Creating a Project” chapter.

Project Navigator Windows

The following figure identifies the various windows / areas included in the Project Navigator.

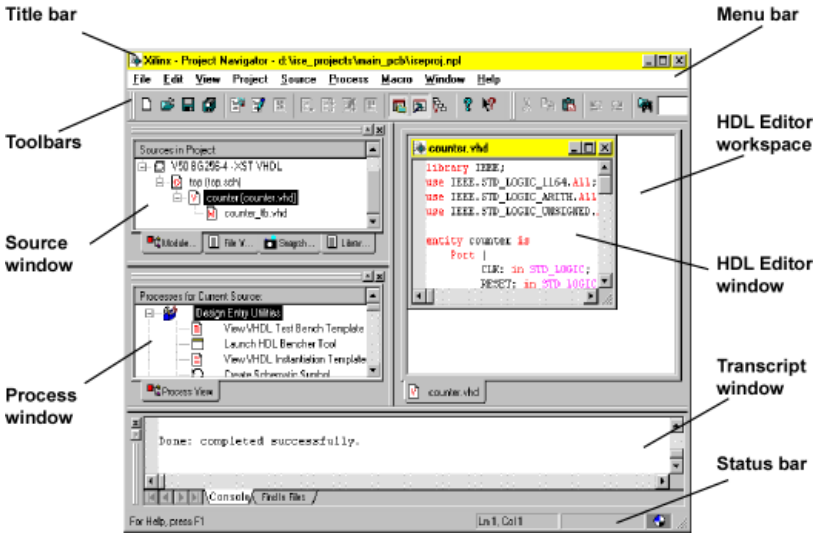


Figure 4-1 Project Navigator Windows

The Project Navigator GUI contains the following sections:

- Source Window

  This window shows all the design files associated with a project. It includes tabs to display the project hierarchy view, files view, snapshot view, and library view.

  In the Source window, user documents are listed above the Device / Synthesis Tool line. Project sources that are processed by the Project Navigator are shown below the Device / Synthesis Tool line.

- Process Window

  This window is in the middle left portion of the Project Navigator and shows the available processes for the selected source. No processes are available for user documents. The processes available for other sources are dependent upon the device and synthesis tool selected for the project as well as the type of source it is.

- Project Workspace

  The Project Workspace consists of the Source window and the Process window. These windows are grouped together for viewing purposes. You can use the **`Project Workspace`** selection on the Project Navigator **`View`** menu to quickly toggle the display of both the Source and Process window together as one item.

- HDL Editor Workspace

  The area on the right side of the Project Manager is the HDL Editor workspace. HDL files are created and edited in this area. Text files can also be created and edited in this area.

- Transcript Window

  The Transcript window at the bottom of the Project Navigator displays informational, warning, and error messages. An error/warning Navigation feature is included within this window to help debug your design.

Note The Schematic Editor and Report Viewer appear in their own windows separate from the Project Navigator window configuration.

Source Window

The Project Navigator Source window (labeled “Sources in Project”) shows all the design files associated with a project. A source is any element that contains information about a design. Sources include the files necessary to describe the behavior of your design (schematics or HDL source files), files needed to test your design (test benches, test fixtures, waveforms for simulation), and general project design documentation. Icons to the left of the source filename identify the type of design source.

Use the **Project** menu in the Project Navigator menu bar to create, add, and copy project sources for your project. The Sources are created and/or added to the project as described in the “Creating/Adding Source Files” section of the “Creating a Project” chapter. Only after a source is added to the project does it appear in the Source window.

Use the **Source** menu in the Project Navigator menu bar to manipulate the sources shown in the Source window. Click on 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, or Move to Library (HDL design files only). You can access the Project Properties dialog box to modify the project title and device/synthesis tool by clicking on the title or device/synthesis tool line in the Source window and then selecting **Properties** from the Source menu.

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

Tabs at the bottom of the Source window select four different views of the source data for your project: Module View, File View, Snapshot View, or Library View.

Module View

Select the **Module View** tab in the Source window to get 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 are listed above the Device / Synthesis Tool line. User documents do not have any processes associated with them. The Process window is blank when a user document is selected in the Source window. An example is shown in the following figure.

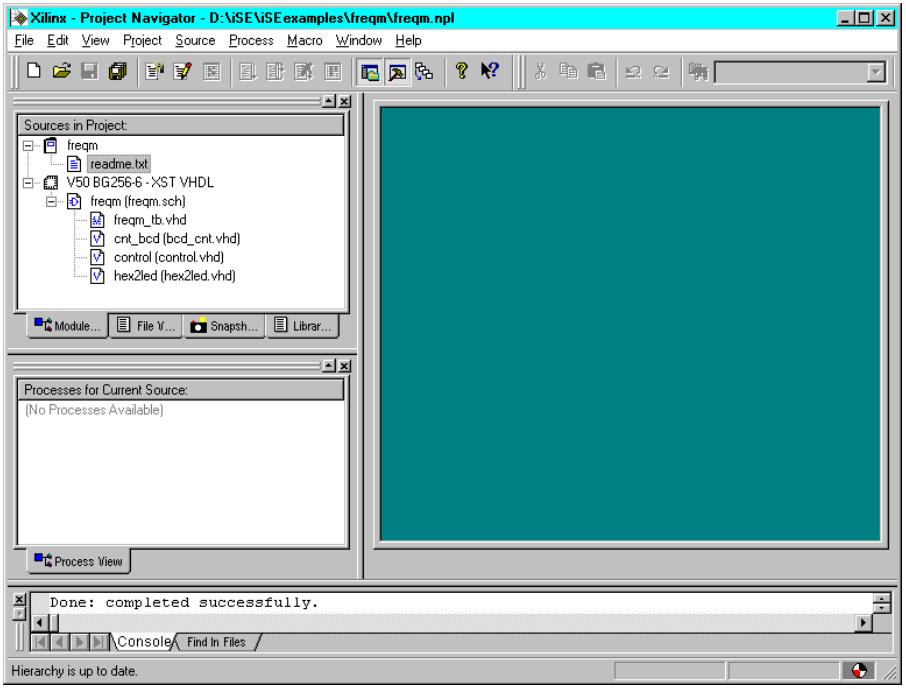


Figure 4-2 User Document Source

Project sources are listed below the Device / Synthesis Tool line in a manner that depicts the relationship of these sources to each other. If you select the Device / Synthesis Tool line or any of the sources below it, the processes the Project Navigator associates with the selected source type are listed in the Process window. The following figure shows an example of processes that may appear for a project source.

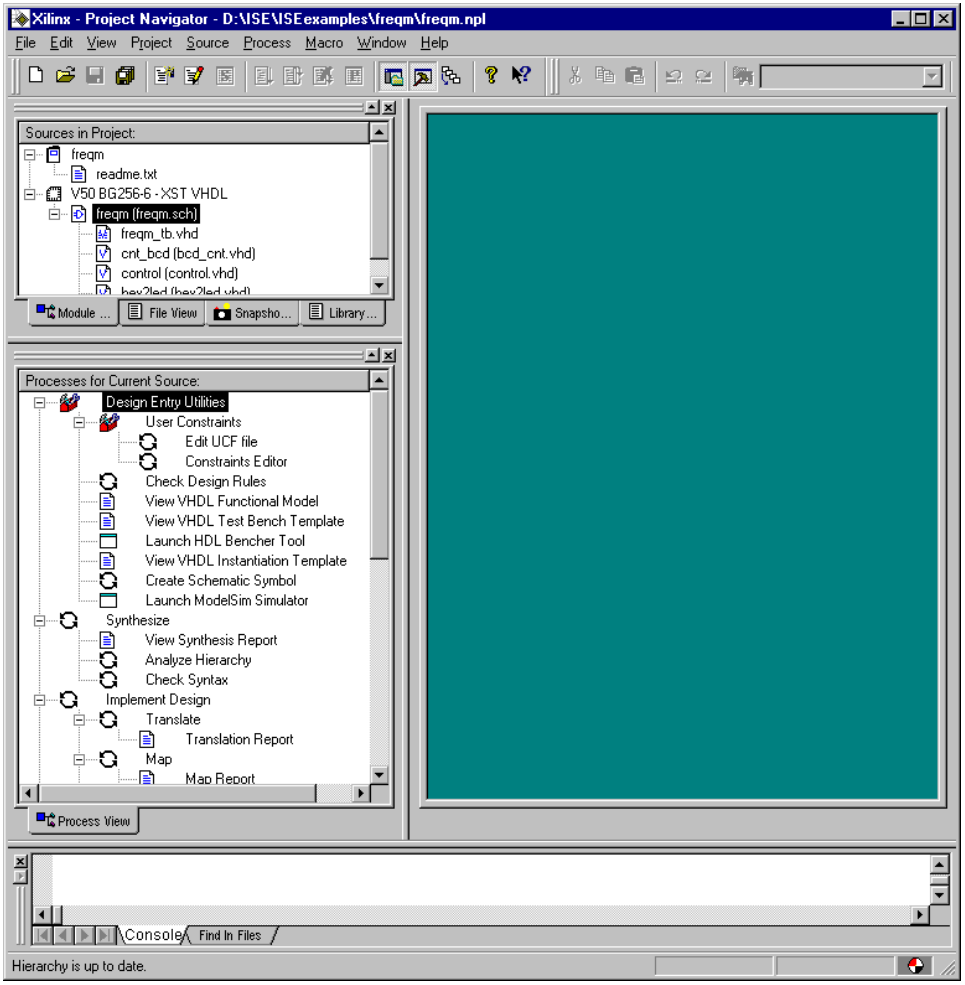
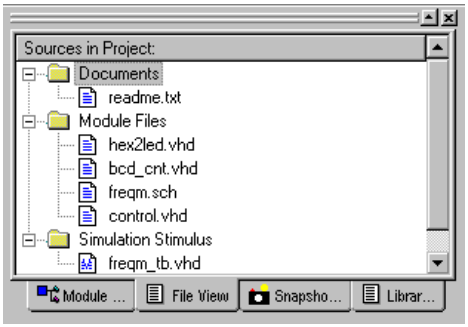


Figure 4-3 Project Source

Both groups (project sources and user documents) fundamentally behave in a consistent manner within the Source window. Double clicking on a source or user document invokes the appropriate editor to view and edit the source or user document. If the source is a schematic, the associated editor is the ECS schematic editor. If a VHDL source file is selected, the appropriate 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.

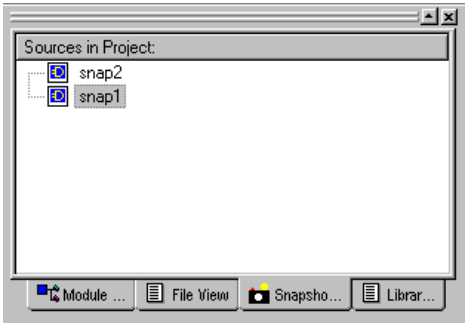
File View

Select the **File View** tab in the Source window to get a file view of the project's sources. The File View displays all of the files in the project so you can manipulate the sources as files rather than as individual modules. The sources are divided by type and listed alphabetically under each type. The following figure shows an example File view.

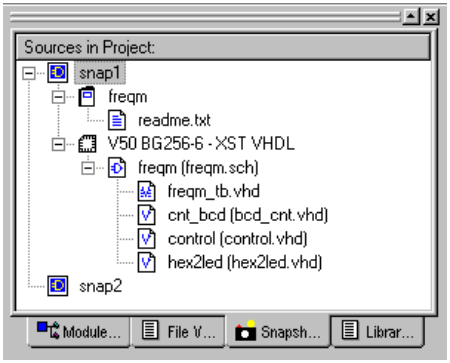


Snapshot View

Select the **Snapshot View** tab in the Source window to view a list of the snapshots you took to preserve versions of your design. Use selections in the Source menu to open, rename, or remove a snapshot. The following figure shows an example Snapshot view.



The Snapshot View allows you to see the contents of each snapshot. To view the contents of the snapshot, click on a snapshot in the Source window and then select **Source** → **Open** from the Project Navigator menu bar.



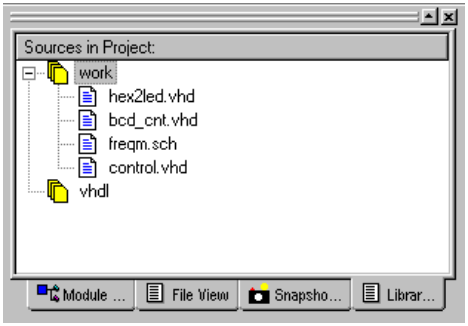
When a snapshot is opened, it is read-only. You can view its elements including generated reports. For example, to view report contents for a source, open the snapshot, click on a design source in the Source window, and then click on a report in the Process window to open it in the Report Viewer.

If you want to perform an operation on a snapshot's elements, you must replace the current project with the snapshot project. Click on a snapshot in the Snapshot view and then select **`Project`** → **`Replace with Snapshot`** from the Project Navigator menu bar. You are prompted to save the current project as a snapshot before the selected snapshot replaces it as the current project.

Refer to the "Snapshots" chapter for complete information on taking and using snapshots.

Library View

Select the **`Library View`** tab in the Source window to view a list of the Libraries associated with the project and the sources included in each one. VHDL projects can include multiple libraries. Refer to the "HDL Library Mapping" chapter for information on creating named VHDL libraries and moving modules to a library. The following figure shows an example Library view.



Source Properties

In the Source window, only the project title and device / synthesis tool sources have properties associated with them. You can change the project title or device / flow selection for a project by highlighting the title or the device / flow icon in the Source window and then selecting **`Source`** → **`Properties`** from the Project Navigator menu bar.

Process Window

The Process window (labeled “Processes for Current Source”) in the middle of the left side of the Project Navigator shows all the processing tasks that apply to whatever object or file is highlighted in the Source window. Processing tasks include such functions as: netlisting, compiling, logic reduction, logic synthesis, place and route, simulation and model-building. In other words, it includes any step along the way from design entry to downloading the design to the device.

The context sensitive capability of the Project Navigator automatically determines the design flow options for a source based on the targeted device and synthesis tool. The Project Navigator displays only those processes that can be performed on a specific source. This reduces the complexity of the user interface by reducing the number of available design process options. In general, the design flows are organized (top to bottom in the Processes window) in logical design sequence.

When you click on a source in the Source window, a list of processing tasks applicable to that source appear in the Process window. In

general, the tasks are listed (top to bottom) in logical design sequence.

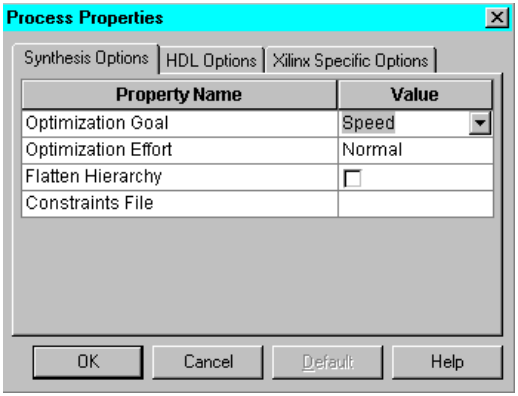
Auto-Make

Project Navigator includes an *auto-make* feature that provides "results-oriented processing." You determine the end result of the processing by selecting an available option in the Processes window. When you click on 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. This feature 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 basically 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, highlight the process and then select **`Process`** → **`Properties`**. (If no properties can be set for that process, the Properties selection is grayed out.) A Process Properties dialog box appears containing parameters appropriate for the process. The Process Properties dialog box for the Synthesis process in an XST-VHDL flow is shown in the following figure.



Viewing Reports

In many cases, a report is the output of a process. You can view a report by double clicking on its name in the "Processes" window or by clicking on its name and then selecting **Process** → **View**.

If the report does not currently exist, it is generated. If a green check mark is in front of the report name, the report is up-to-date and no processing is performed. If the desired report is not up-to-date, you can click on the report name and then select **Process** → **Run** to update the report before you view it. The auto-make process automatically runs only the necessary processes to update the report before displaying it. Or, you can select **Process** → **Run 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. Refer to "Implementing the Design" chapter for descriptions of the implementation reports.

HDL Editor Workspace

The HDL Editor workspace (shown in Figure 4-4) is the main text editing area for HDL code. It is the only window of the Project Navigator that cannot be hidden or undocked (refer to the "Docking / Undocking Project Navigator Windows" section).

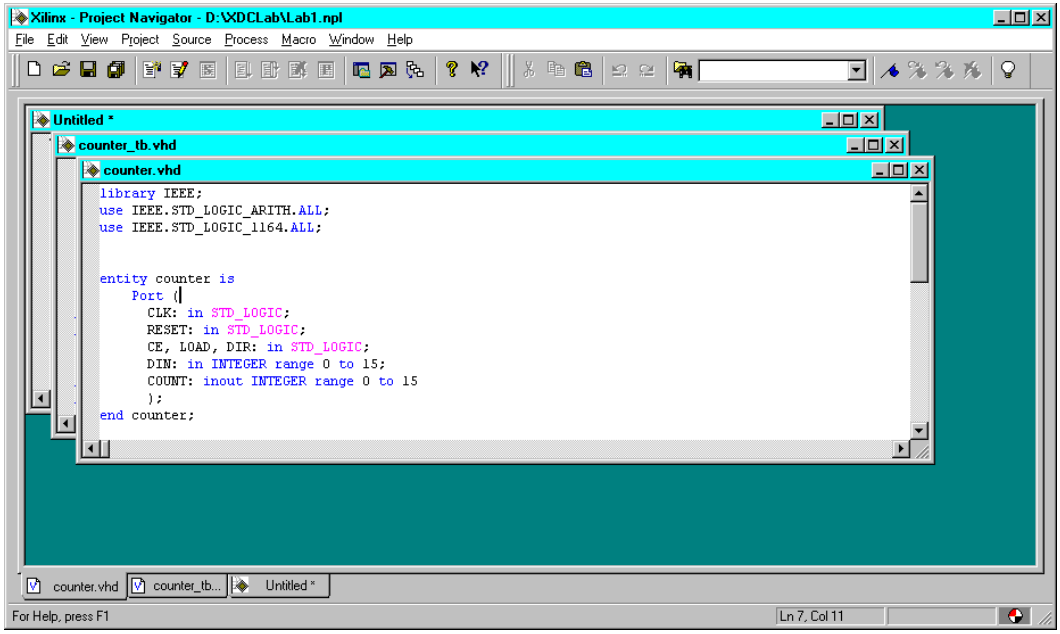


Figure 4-4 Project Navigator HDL Editor Workspace

Tabs at the bottom of the HDL Editor workspace allow easy navigation to the file you want to view or edit. The Window menu on the Project Navigator provides the usual window functions to manage the various windows open within the HDL Editor Workspace.

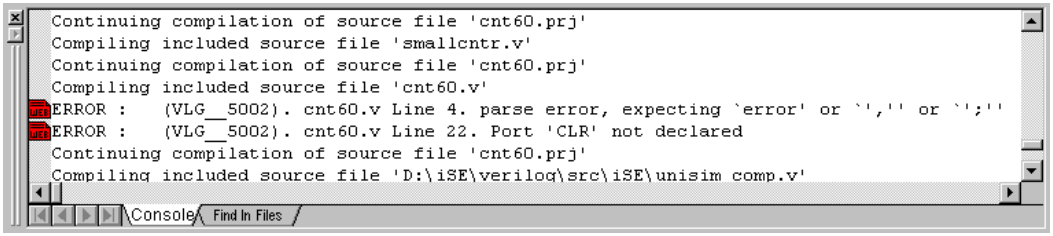
The File, Edit, and Macro menu on the Project Navigator are used with the HDL Editor workspace. Selections on these menus are discussed more fully in the “HDL Sources” chapter.

Two toolbars are available in the Project Navigator: the Standard toolbar (left side of toolbar area) and the Editor toolbar (right side of the toolbar area). The Editor toolbar (shown undocked below) is used exclusively with the files in the HDL Editor workspace. Refer to the online help for information on using the toolbars.

Error Navigation from the Transcript Window

The Transcript window at the bottom of the Project Navigator main window contains a project log. The output of all programs is captured here, including all error / warning messages from the synthesis and implementation tools. These errors and warnings are "navigable." That is, you can navigate to a line in the source file containing the error or to a Solution Record for the problem.

An example of a Transcript window with the red error icons that represent source / solution record navigable errors is shown in the following figure. (The icon representing errors that are navigable to a solution record only is similar; it does not have the two lines above the word "WEB." The corresponding warning icons are yellow instead of red.)



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. To navigate to a source file, do the following:

1. Place your cursor in the Transcript window.
2. Scroll to a line of text containing an error or warning message
3. Double-click on the error / warning icon in the Transcript window. (Or, right-click on the error / warning icon and then select **`Goto Source`** from the menu that appears.)
4. The source file opens to the line containing the error. See the following figure for an example.

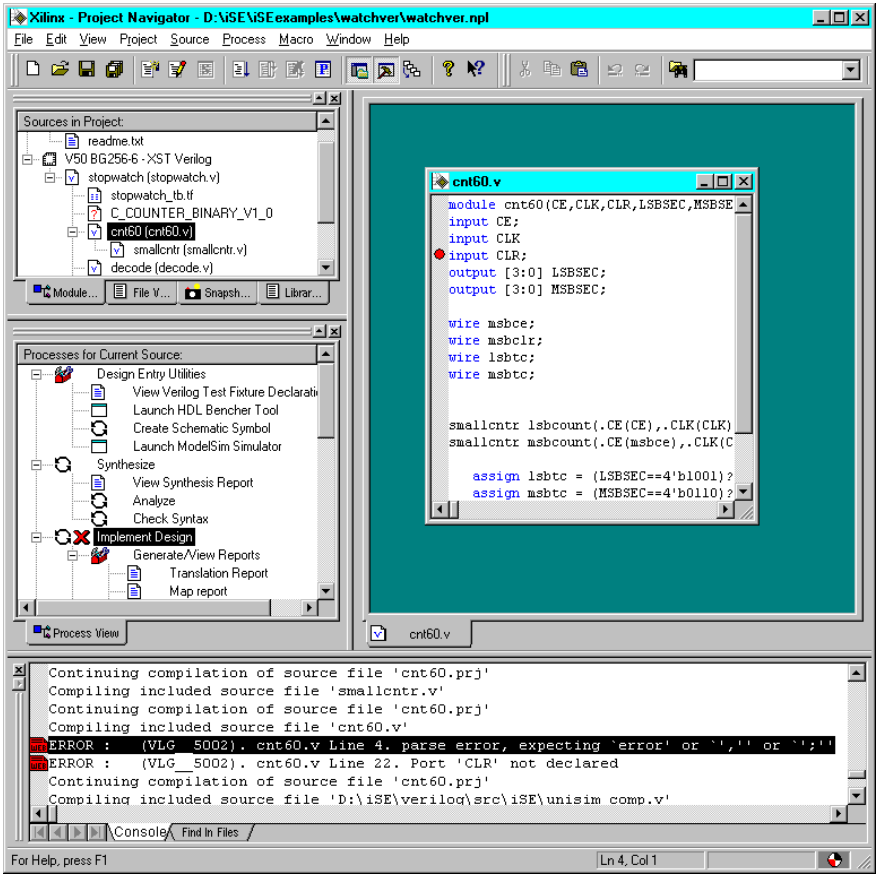


Figure 4-5 Example of Error Navigation from the Transcript Window to the Source File

For all errors and warnings, you can go to the Xilinx website where you can use the Solution Record search engine to find a Solution Record pertaining to your problem. To navigate to a Solution Record on the Xilinx website, do the following:

1. Place your cursor in the Transcript window.
2. Scroll to a line of text containing an error or warning message.
3. Right-click on the icon in front of the line containing the warning or error.
4. Select **`Goto Solution Record`** from the menu that appears.

5. Selecting this option, takes you to the Solution Record search engine on the Xilinx website.

Error navigation is available as processes are running. You do not need to wait for a process to stop to navigate to displayed errors in the Transcript window.

Note Errors and warnings that appear in the report files are not captured in the Transcript window and, therefore, are not navigable.

Integrated Tools

The Project Navigator includes integrated support for the following tools so that you can use them with your Foundation Series ISE projects:

- StateCAD and StateBench from Visual Software Solutions, Inc. for state machine creation

  Refer to “State Diagrams” chapter for information on using these tools with Foundation Series ISE.

- ECS Schematic Editor from Xilinx for schematic and symbol creation and editing

  Refer to the “Schematic Sources” chapter for information on the Schematic and Symbol editors.

- ModelSim XE / PE / SE from Model Technology Inc. for simulation

  The “Simulation” chapter describes the integration of ModelSim within the Project Navigator.

- HDL Bencher from Visual Software Solutions, Inc. for automated testbench / test fixture creation

  The “Simulation” chapter also describes the integration of the HDL Bencher within the Project Navigator.

- XST for synthesis

  Refer to the “Synthesis” chapter for information on how XST is used with Foundation Series ISE.

- FPGA Editor from Synopsys for synthesis

  Refer to the “Synthesis” chapter for information on how FPGA Express is used with Foundation Series ISE.

- ISE Report Viewer for viewing implementation reports and other reports. Refer to the "ISE Report Viewer" section of the "Implementing the Design" chapter for information on this tool.

Customizing the Project Navigator

You can set many environment variables and change settings for the Project Navigator. Changes can be made as described in the following sections.

Setting Display Preferences

Select **`Edit`** → **`Preferences`** from the Project Navigator menu to access the Preference dialog box (shown below).

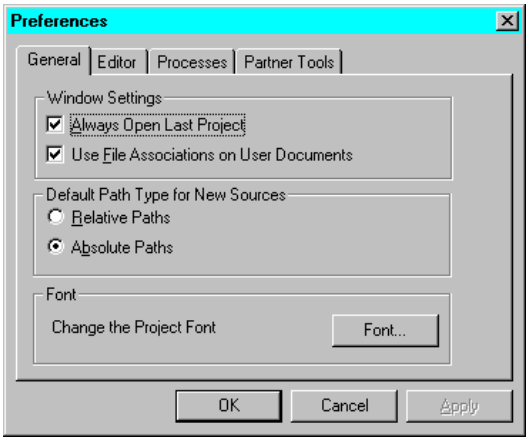


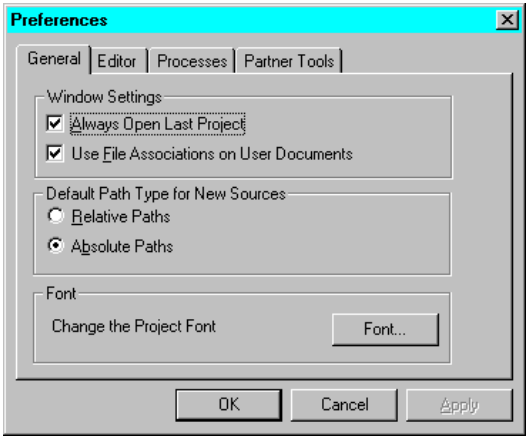
Figure 4-6 Project Navigator Preferences Dialog Box

General Preferences

From the General tab (see Figure 4-6) you can set general window settings, the default path for new sources, and the project font (the font used in the Source and Process window).

Editor Preferences

From the Editor tab, you can select attributes for tabs and the font used in HDL Editor windows.

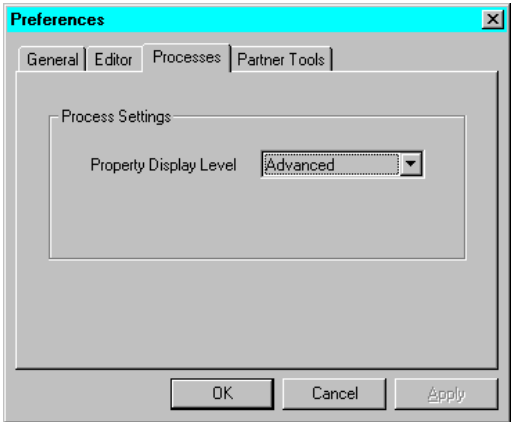


Standard/Advanced Process Properties Preference

In the Project Navigator, you may 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.

Use the following procedure to include advanced properties in the Process Properties lists.

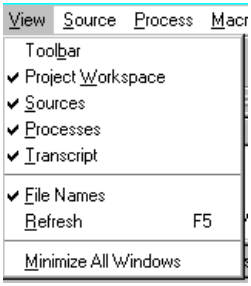
1. Select **`Edit`** → **`Preferences`** from the Project Navigator menu.
2. Click the **`Processes`** tab (shown in the following figure).



3. Click in the Property Display Level box and select **Advanced** from the pull-down menu that appears.

Displaying/Hiding Windows and Toolbars

Click **View** on the Project Navigator to display a list of items (shown below) that you can use to control which windows you want displayed in the Project Navigator.



Docking/Undocking Project Navigator Windows

Project Navigator includes a docking / undocking feature for most of its windows and toolbars. This allows you to “remove” windows and toolbars from the default Project Manager window and place and size them separately for convenience of use. For example, Figure 4-7 represents the default configuration for the various Project Navigator windows / toolbars.

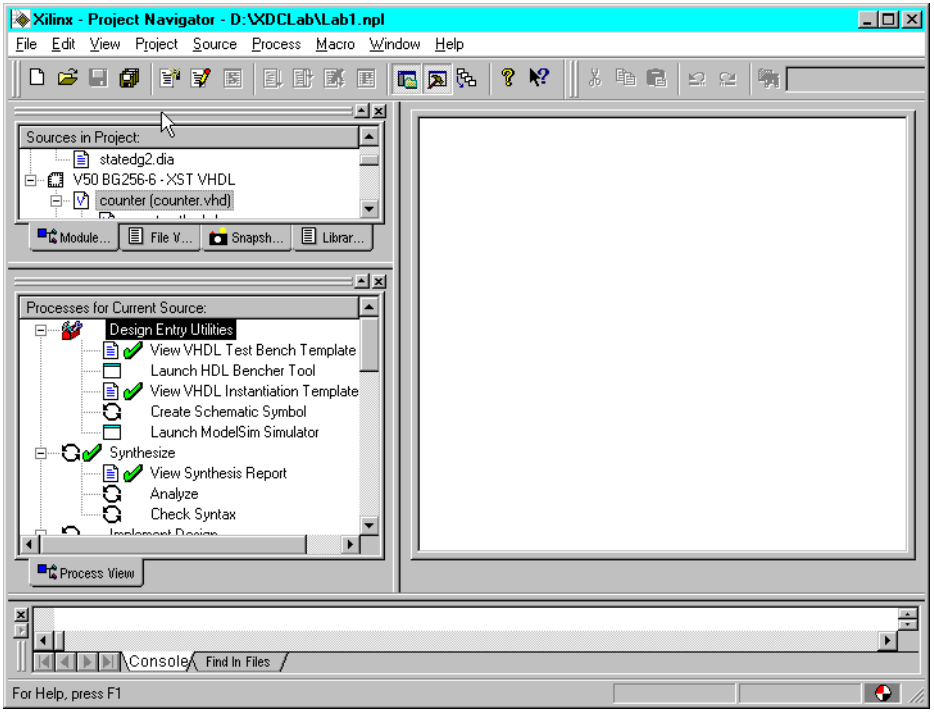


Figure 4-7 Default Project Navigator Window Configuration

You use “grabber” bars to undock a window or toolbar from the Project Navigator. The grabber bars are the two parallel bars on the top or side of the window or toolbar. The arrow in Figure 4-7 points to the grabber bars for the Source window.

You can use any of the following methods to undock toolbars or windows from the Project Navigator.

- Double-click on the grabber bars at the top or side of a window or toolbar. The window or toolbar is immediately undocked from Project Navigator. You can now move and place it as desired.
- Hold down the left mouse button over the grabber bars and drag the window or toolbar to the desired position.
- Right-click on the grabber bar and uncheck **`Allow Docking`** on the menu that appears.

Figure 4-8 shows an example of the Source window and Process window undocked from the Project Navigator. The Source and Process windows are shown below the Project Navigator, but they can be moved anywhere.

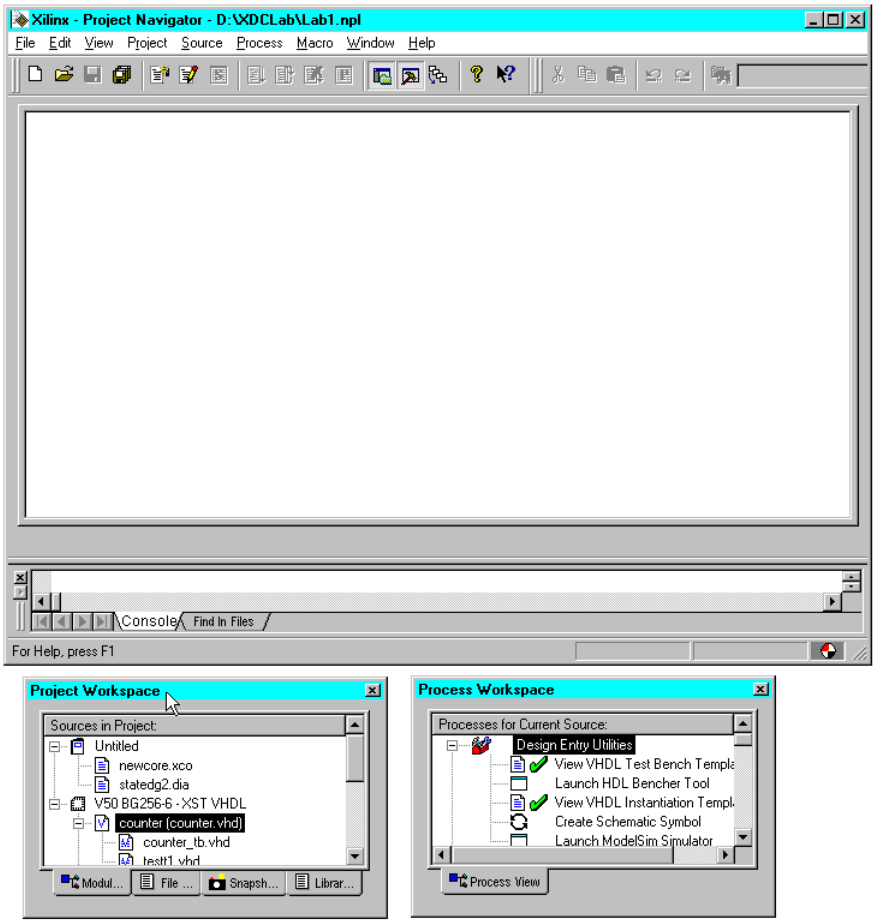
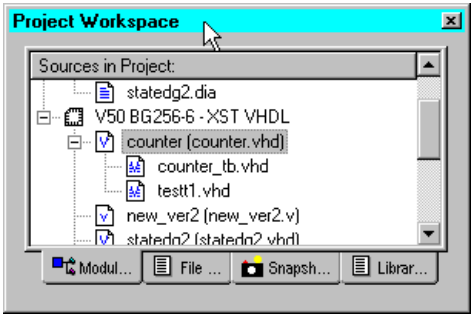


Figure 4-8 Example Project Navigator Window Configuration with Undocked Source and Process Windows

The window can be redocked using either of the following methods.

- Double-click on the window title bar (indicated by the arrow in the following figure.) The window is immediately redocked to the Project Navigator.



- Drag the window to the bottom area of the Project Navigator main window until you see the drag shape change to the docked shape.

Figure 4-9 shows the Source window redocked to the Project Navigator.

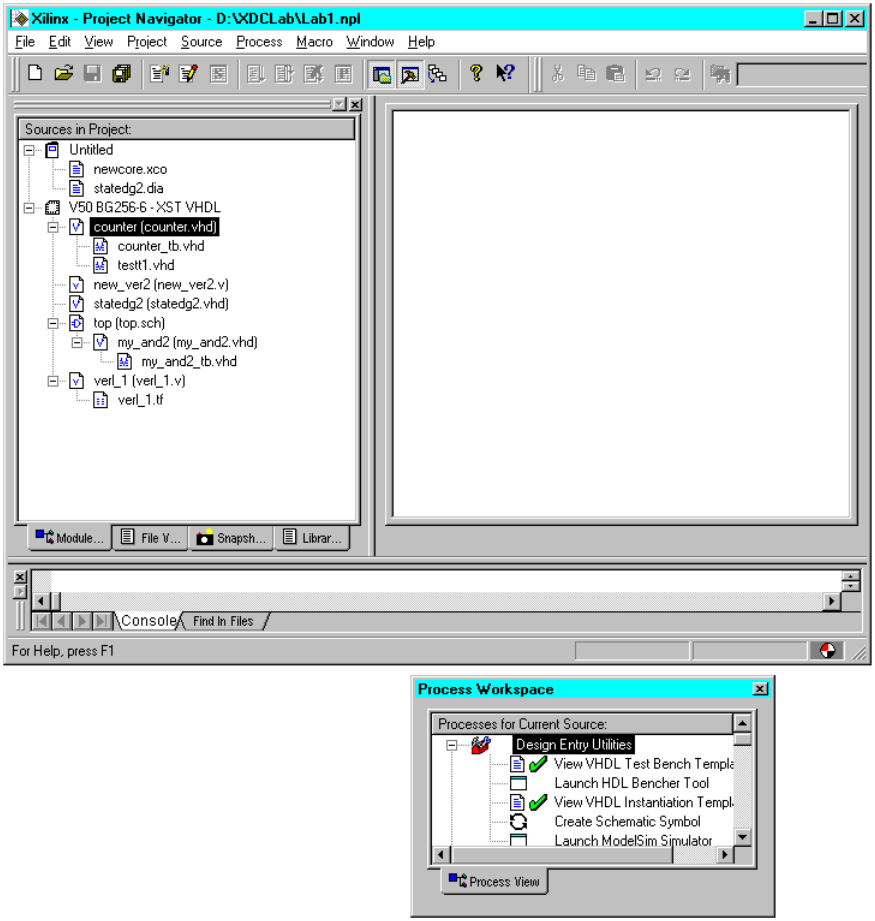
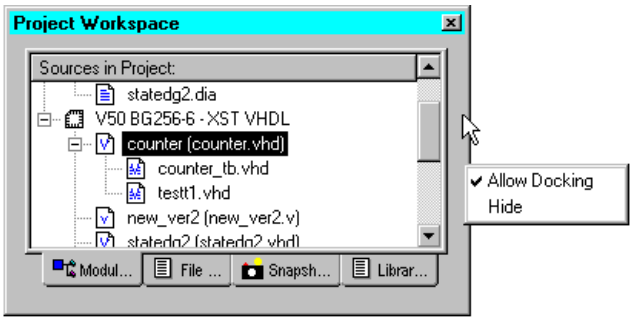


Figure 4-9 Example Project Navigator Window Configuration with Redocked Source Window and Undocked Process Windows

By default, all windows with grabber bars allow docking to the Project Navigator. If you want to be able to move a window or toolbar around without having the Project Navigator redock whenever it moves over it, you can disallow docking for that window or toolbar using one of the following procedures.

If the window or toolbar is currently docked to the Project Navigator, right-click the “grabber” bars. Then click on the **Allow Docking** selection in the menu that appears until the check mark is removed.

If the window is already undocked from the Project Manager, right-click in its right, left, or bottom window border (not on the title bar). Then click on the **Allow Docking** selection in the menu that appears (shown in the following figure) until the check mark is removed.



Note You can use the **Hide** selection (below the **Allow Docking** selection) to hide the window or toolbar. To redisplay it, select the appropriate window or toolbar from the Project Navigator **View** menu.

Chapter 5

HDL Sources

You can create your design using HDL code only or a combination of HDL code, schematics, and state diagrams. This chapter describes the creation of HDL design sources and contains the following sections:

- "Supported Languages"
- "Creating HDL Source Files"
- "Opening HDL Source Files"
- "HDL Editor"
- "Language Templates"
- "Creating a Schematic Symbol from an HDL Source"

Supported Languages

ISE supports the following languages for the creation of HDL source files: VHDL, Verilog, and ABEL-HDL. The ABEL-HDL language is only supported for CPLD designs. The languages that can be included in your project depend on the targeted device and the synthesis tool selection.

VHDL

VHSIC (VHSIC is an acronym for Very High-Speed Integrated Circuits) Hardware Description Language. An industry-standard (IEEE 1076.1) HDL. Recognizable as a file with a .vhd or .vhdl extension.

VHDL can be used to model a digital system at many levels of abstraction ranging from the algorithmic level to the gate level. It is IEEE standard 1076-1993.

VHDL is capable of describing the concurrent and sequential behavior of a digital system with or without timing.

Refer to the *XST User Guide* for specific information on using VHDL in projects with the XST synthesis tool. Refer to the *Synthesis and Simulation Design Guide* for specific information on using VHDL in projects with the FPGA Express synthesis tool.

Verilog

Verilog is an industry-standard HDL (IEEE Std 1364) originally developed by Cadence Design Systems, now maintained by OVI. Recognizable as a file with a .v extension.

Verilog is a commonly used Hardware Description Language (HDL) that can be used to model a digital system at many levels of abstraction ranging from the algorithmic level to the gate level. It is IEEE standard 1364-1995.

Refer to the *XST User Guide* for specific information on using Verilog in projects with the XST synthesis tool. Refer to the *Synthesis and Simulation Design Guide* for specific information on using Verilog in projects with the FPGA Express synthesis tool.

ABEL-HDL

ABEL is a high-level language (HDL) and compilation system.

In Foundation Series ISE, ABEL-HDL is supported for CPLD devices only and only with the ABEL XST and ABEL BLIF synthesis tools. It is not supported for FPGA devices or for CPLDs devices used with a synthesis tool 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. You can access an HDL Converter by selecting the Device / Synthesis Tool line in the Source window. Right-click on HDL Converter (under Design Utilities) in the Process window. In the Project Properties dialog, enter the name of the ABEL file you want to convert and select whether to convert it to VHDL or Verilog.

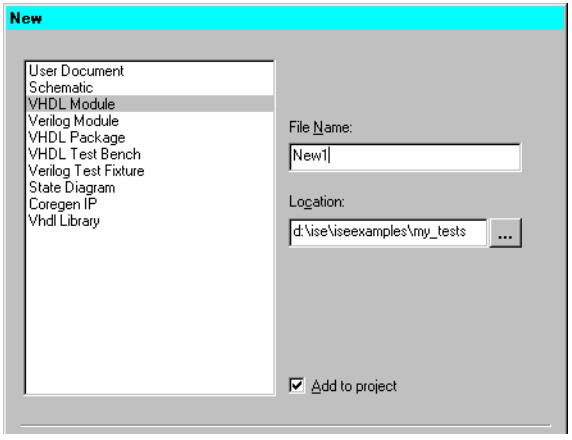
Creating HDL Source Files

You can use the text editor of your choice to create an HDL source file and then add that file to your project. Or, you can use Project Navigator's New source wizard and HDL Editor. With the New source wizard, you are presented with a series of dialog boxes where you enter information about the new source. The Project Navigator uses this information to create and open a skeleton file in the selected language in its HDL Editor. You select whether you want the file added to the project.

New HDL Source Wizard

Following is the procedure used to create an HDL module with the New source wizard. The procedure described below is for creating a VHDL file. The procedure for creating Verilog or ABEL-HDL files is similar to this procedure.

1. Open or create your project (see the "Creating a Project" chapter).
2. Select **`Project`** → **`New Source`** from the Project Navigator to access the New source window.
3. Select the HDL file type you want to create from the list of available source types displayed in the New source window. The HDL source types included in the list depend on the device and synthesis tool you selected for your project. The following figure represents the available sources for a Virtex FPGA Express flow. An XST flow would not include both VHDL and Verilog choices.



4. Enter a name for the new HDL file in the File Name box. Refer to the “Creating / Adding Source Files” section of the “Creating a Project” chapter for detailed information on the New source screen.
5. Click **Next** when you are ready to proceed.
6. For new VHDL modules, the Define VHDL Source window appears. (Similarly, the Define Verilog Source window or Define ABEL-HDL source window appears if you selected to create a Verilog or ABEL-HDL module, respectively.) You can use this window to create skeleton code for the VHDL module you are describing.

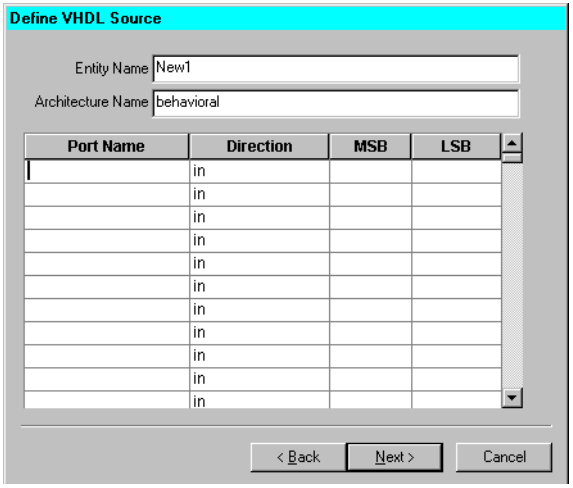
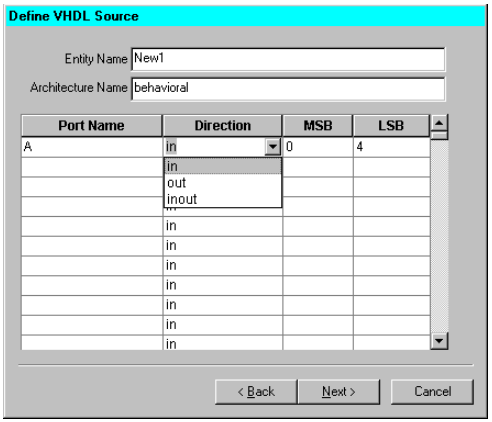


Figure 5-1 Define VHDL Source Dialog

7. Enter a Port Name. Click in the right side of the Direction box. Then use the pull down menu that appears to select a direction (in, out, inout).

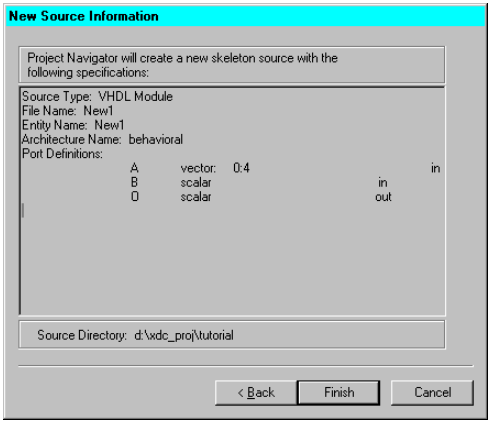


Click in the right side of the Most Significant Bit (MSB) box and Least Significant Bit (LSB) box to access their selector arrows. Use the up and down arrows to select the desired value.

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

Click **`Next`** when you are ready to continue.

8. The New Source Information window appears with a summary of the specifications made in the Define VHDL Source window.



Click **`Finish`** to proceed.

9. The HDL Editor opens in the Project Navigator's workspace with the newly created skeleton code displayed in it.

   If you selected to have this file added to the project, the new file is automatically added to the project as indicated in the Source in Project window.

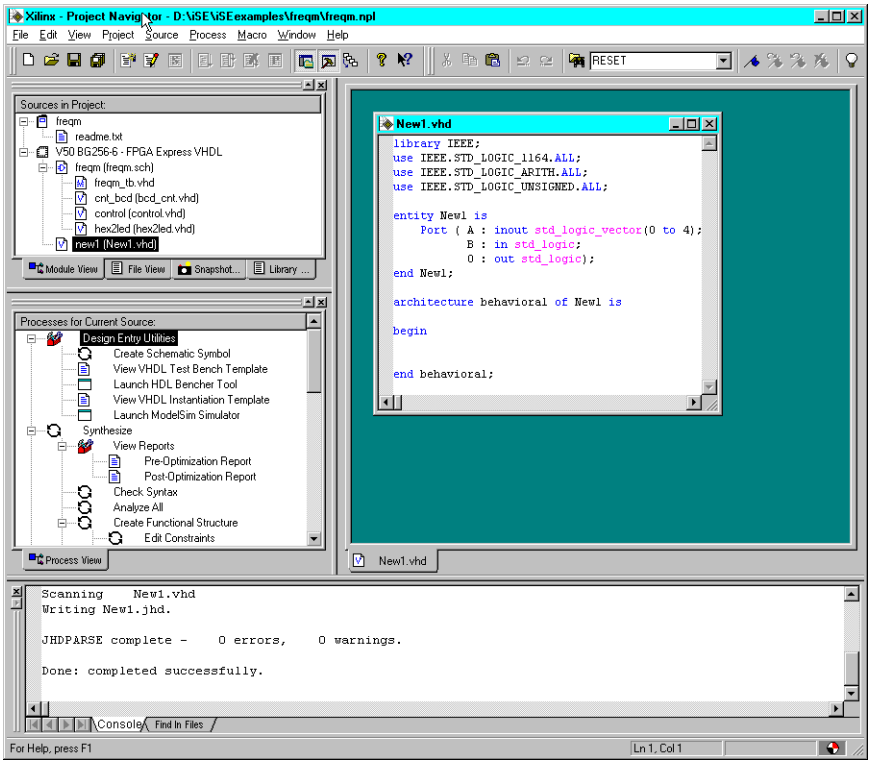


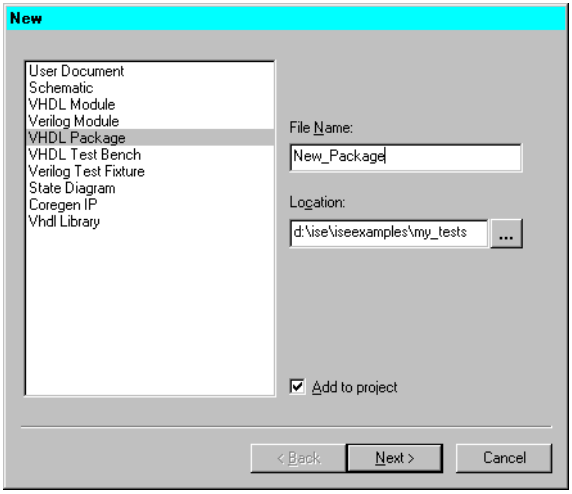
Figure 5-2 HDL Editor with New1.vhd File

10. You can now use the HDL Editor to continue coding from the new module.

Skeleton Code for New VHDL Testbenches, VHDL Packages, or Verilog Templates

Following is the procedure used to create skeleton code for a new VHDL package. The procedure to create skeleton code for a new VHDL testbench or Verilog test fixture is similar. The difference is that you would select **`VHDL Testbench`** or **`Verilog Test Fixture`** and the skeleton code produced would be for a VHDL testbench or Verilog test Fixture.

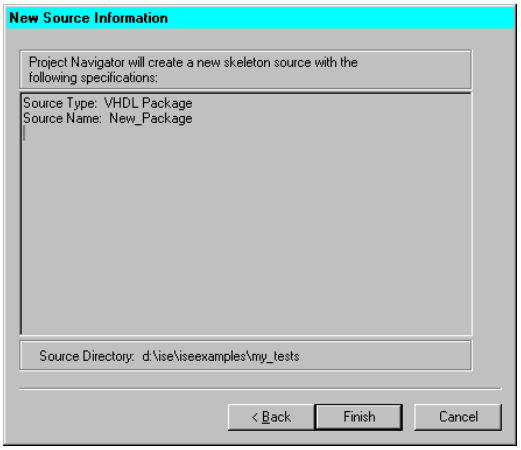
1. Open or create your project (see the "Creating a Project" chapter).
2. Select **`Project`** → **`New Source`** from the Project Navigator to access the New source window.
3. Select **`VHDL Package`** from the list of available source types displayed in the New source window.



4. Enter a name for the new VHDL package in the File Name box. Refer to the "Creating / Adding Source Files" section of the "Creating a Project" chapter for detailed information on the New source screen.

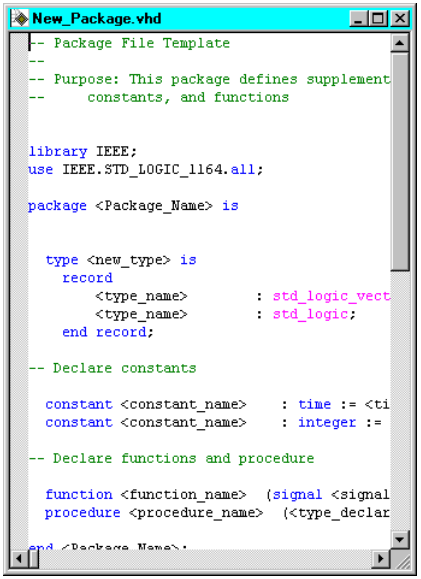
   Click **`Next`** when you are ready to proceed.

5. The New Source Information window appears with a summary of your request. Click **`Finish`** to continue.



6. The HDL Editor opens in the Project Navigator's workspace with skeleton code for a new VHDL package.

   If you selected to have this file added to the project, the new file is automatically added to the project as indicated in the Source in Project window.



7. You can now 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, simply double click on its name. The file then appears in the HDL Editor window in the Project Navigator. The HDL Editor is language sensitive and identifies the language in the file by the file extension.

HDL Editor

Foundation Series ISE provides a general purpose text editor that is HDL language sensitive. You can invoke the HDL Editor in any of the following ways:

- Select **Project** → **New Source** from the Project Navigator menu. Follow the new source creation sequence. At the end of the sequence, the newly created HDL file opens in the HDL Editor.
- Double click on any HDL file listed in the Source window.
- Select **File** → **New** from the Project Navigator menu.

  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 Online Help

Detailed procedures and commands for using the HDL editor can be found in the HDL Editor online help. Context sensitive online help, especially for reserved words, is available in open HDL files by highlighting a word or phrase and then pressing **F1**.

Use the following procedure to access comprehensive HDL Editor online help:

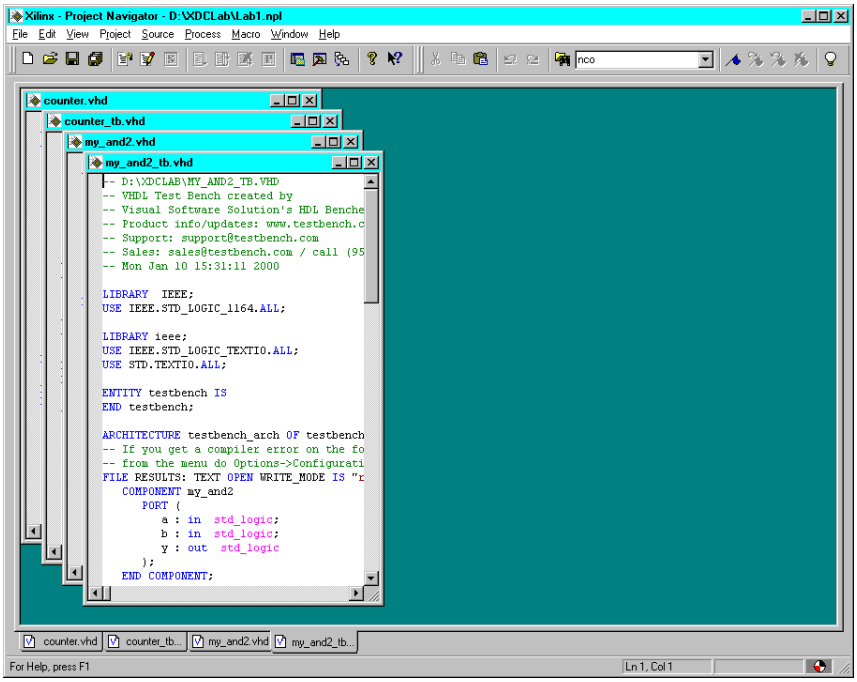
1. In the Project Navigator menu, click **Help** → **Foundation ISE Help Contents** to display the Xilinx Foundation Series ISE On-line Help System menu (umbrella help menu).
2. Click **HDL Editor** under Design Entry in the Xilinx Foundation Series ISE On-line Help System menu to open the HDL Editor help topics window.

File/Window Operations

Use the second group of selections in the Project Navigator **File** menu (shown in the following figure) for opening, closing, printing, and saving files in the HDL Editor workspace. The File menu also includes the Recent Files list.

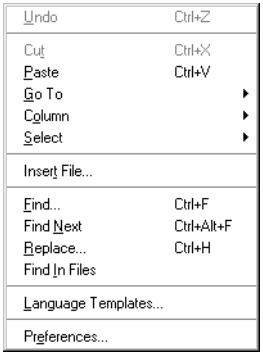
You can have multiple files open at one time. Tabs at the bottom of the HDL Editor window help you move between files.

The easiest way to maximize the HDL Editor workspace is to click the Toggle Workspace Window toolbar button to hide the Source and Process windows. Click on the Tool Transcript View toolbar button to hide the Transcript view. Refer to the “Docking / Undocking Project Navigator Windows” section of the “Project Navigator” chapter for additional information on how to maximize the HDL Editor workspace.

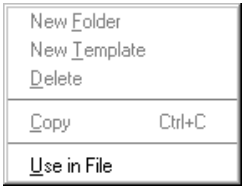


Editing Functions

The Project Navigator Edit menu (shown in the following figure) contains the cursor movement, selection, copy, cut, paste, and insert file functions that you use with the HDL Editor. Icons on the Project Navigator Tool Bar are also available for the basic functions. Refer to the HDL Editor's online help for details.

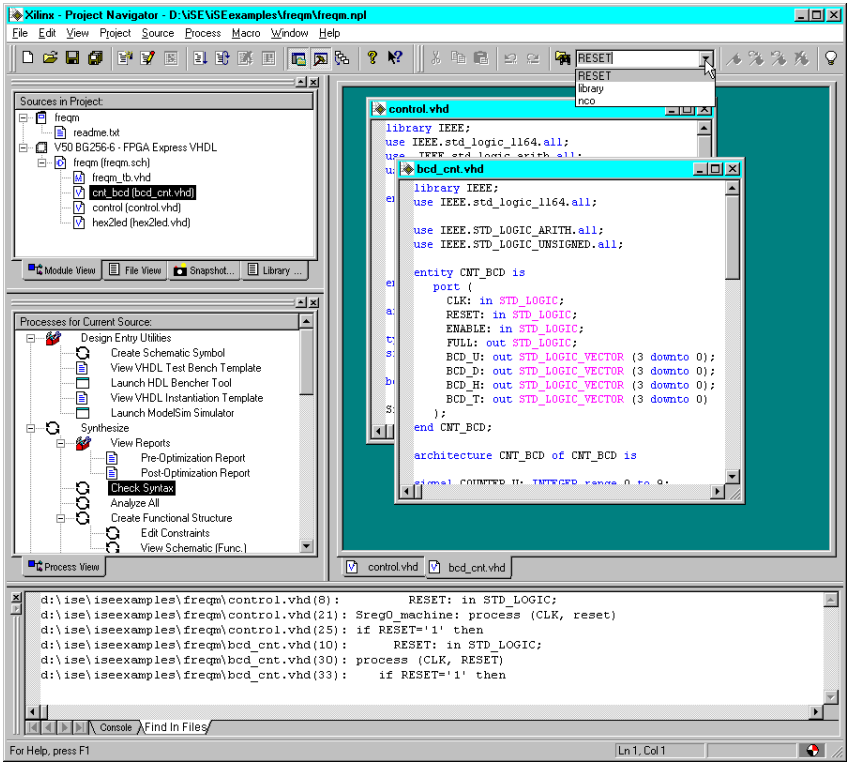


When the Language Template window is active, the Project Navigator Edit menu changes as shown in the following figure to reflect the edit functions available with the Language Templates tool. A separate toolbar on the Language Template window is also available for these functions.



Search Functions

A portion of the Project Navigator's Edit menu is dedicated to 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 tool bar. 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. The following figures illustrates a search for the word "RESET" in two open files.



Macro Functions

The Project Navigator Macro menu (shown below) contains functions you can use for keyboard recording and playback.

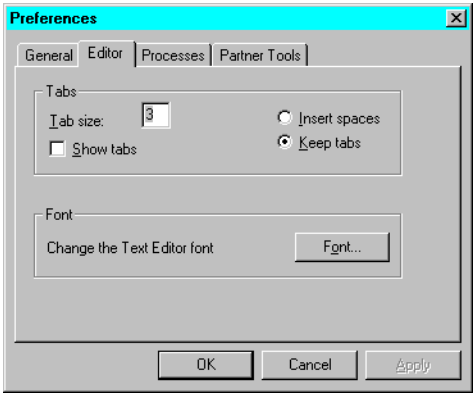
The HDL Editor will allow the user to record his own macros for use in the active document. Macros can be filed in user created folders and recalled in future documents.

Note You are not prompted to save your macros when closing the Project Navigator. If you do not select Save Macros, your recorded macros will not be available for future projects.

Customizing Tabs and Fonts

To customize the tab settings and / or fonts used with the HDL Editor, do the following:

1. Select **Edit** → **Preferences** from the Project Navigator main menu.
2. When the Preferences menu appears, click the **Editor** tab.



3. Use the Tabs and Font areas on the Editor Preferences menu to modify these items as desired.

Note The background color of the HDL Editor workspace is controlled by your PC'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. Refer to Table 3-2 in the "Creating a Project" 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 / reserved word identification as appropriate for the language contained in the file. You can check whether a word in a open HDL file is a reserved word by highlighting it and then pressing **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. Refer to the “Language Templates” section for information on this tool.

Language Templates

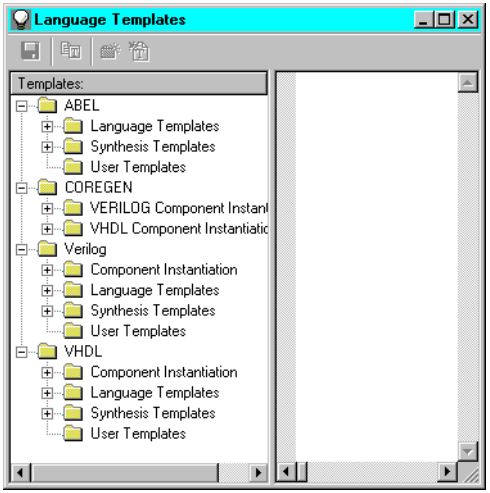
Multiple language and synthesis templates with prepared pieces of code are available in Foundation Series ISE. These enable easy insertion of pre-built text structures such as common language structures or instantiation templates for synthesis into your HDL source file.

Accessing the Language Templates

You can access the Language Templates tool using either of the following methods:

- Select **`Edit`** → **`Language Templates`** from the Project Navigator menu
- Click the Language Template icon (shown in the following figure) in the Editor toolbar

The Language Template window (shown in the following figure) opens in the HDL Editor window.



The Language Templates window consists of two sections. The left side allows you to list the available code templates for ABEL, Verilog, VHDL, and the CORE Generator components. The window on the right displays the template information (text) when you select a template.

Selecting an Existing Template

To select a template for use in your HDL code, do the following:

1. Open the Language Templates window as described in the "Accessing the Language Templates" section
2. Click on the "+" symbol in front of **ABEL**, **COREGEN**, **Verilog**, or **VHDL** to display the folders under each selection (see the figure in the "Accessing the Language Templates" section).

   There are four groups of templates for VHDL and Verilog, three for ABEL, and two for COREGEN. *Language templates* contain basic language constructs. *Synthesis templates* have code fragments for synthesis-oriented implementation of basic functional blocks, such as multiplexers, flip-flops, counters, etc. *Component instantiation templates* are templates for instantiating Xilinx library components and CORE Generator cores into your VHDL and Verilog source code. *User templates* are user created templates for specific constructs.

3. Click on the folder under the selected language or COREGEN that represents the type of template you want. Browse through the underlying folders and files to select the desired template.
4. Click on a template name to display the template (and example code for Language Templates) in the window on the right. The following figures show examples of the three main types of templates: component instantiation templates, language templates, and synthesis templates.

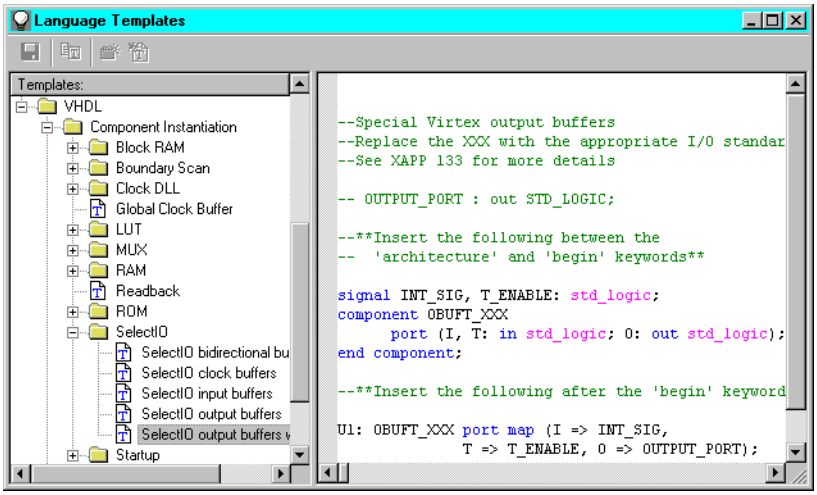


Figure 5-3 Component Instantiation Template Example

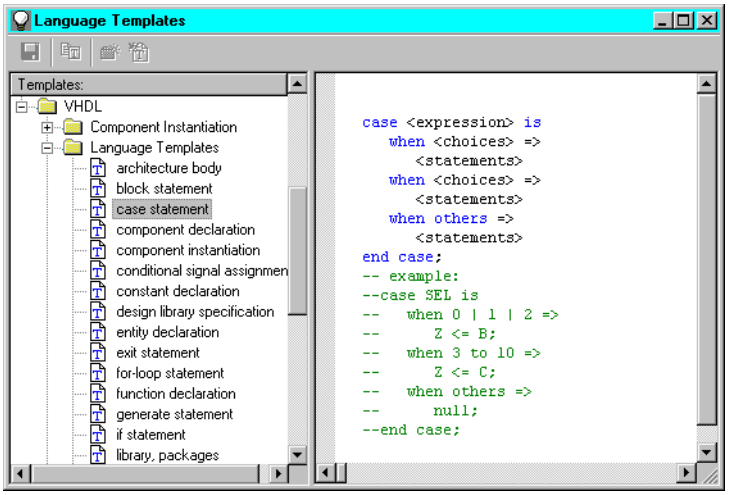


Figure 5-4 Language Template Example

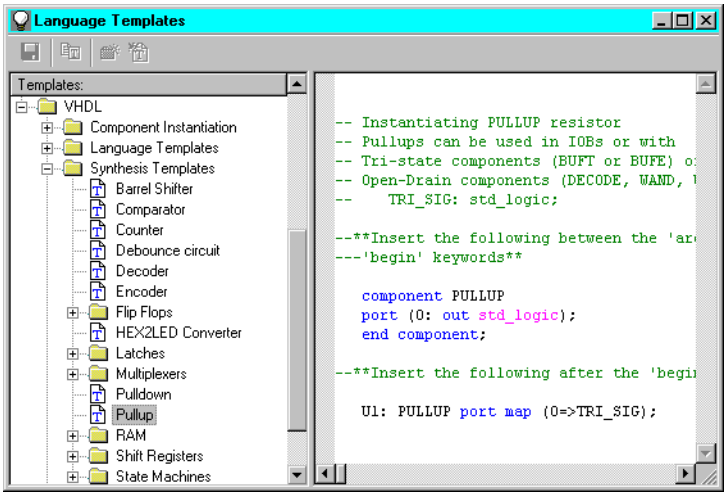


Figure 5-5 Synthesis Template Example

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

- Select a template from the Template area and drag (hold the left mouse button down) and drop it into the HDL code. You can scroll within the HDL Editor to the correct line by continuing to hold the left mouse button down and moving to the top or bottom of the Edit window. A special cursor indicates where the text is going to be dropped.
- Highlight the template text. Select **Use in File** from the Edit menu in the Project Navigator. Place your cursor where you want to insert the text. Right click and then select **Paste** from the menu that appears.
- Highlight the template text. Click on the **Use in File** icon on the Language Templates toolbar. Place your cursor where you want to insert the text. Right click and then select **Paste** from the menu that appears.
- Highlight the text you want to insert and "drag-and-drop" it into your document.

Creating a User Template

To create a new User Template, do the following:

1. Open the Language Templates window as described in the "Accessing the Language Templates" section
2. Select **ABEL**, **Verilog**, or **VHDL** from the Templates window of the Template Assistant.
3. Highlight the User Templates folder under the selected language.
4. Click the **New Template** icon on the Template Assistant toolbar. Or, select **New Template** from the Project Navigator Edit menu.

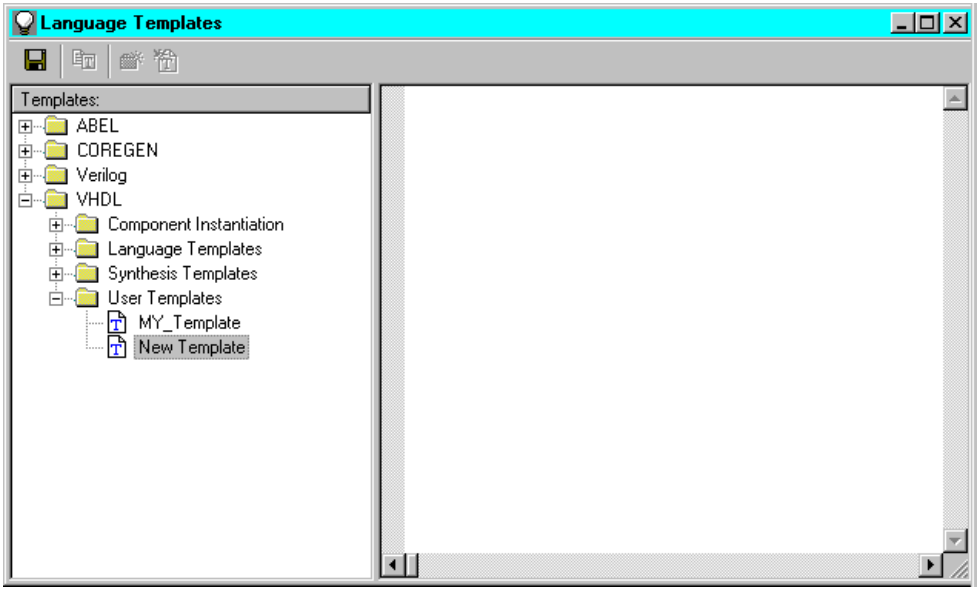


Figure 5-6 New User Template Creation

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.

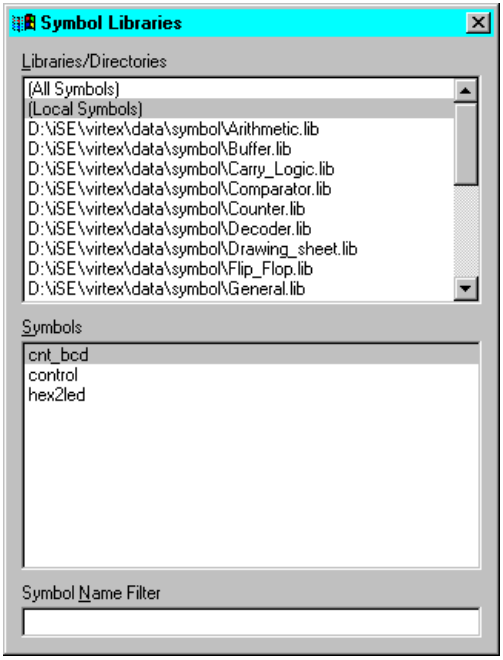
Creating a Schematic Symbol from an HDL Source

To create a schematic symbol from an HDL file for use on a schematic, use the following procedure.

1. In the Sources in Project window, highlight the desired HDL file. The file can contain underlying schematics or other HDL modules.
2. In the Process window, double click the **`Create Schematic Symbol`** process.

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 Schematic Editor's Symbol libraries.

To use the symbol in a schematic, open a schematic and then select **`Add`** → **`Symbol`** from the Schematic Editor menu. Select the symbol from the Local Symbol directory (shown in the following figure) for placement on the schematic. Refer to the "Schematic Sources" chapter for information on the ECS Schematic Editor.



After you add a symbol created from a HDL source, you can view and modify the HDL text by selecting **View** → **Push/Pop** from the Schematic Editor menu and then selecting the HDL source's symbol on the schematic. The HDL source file opens in the HDL Editor workspace.

Chapter 6

Schematic Sources

This chapter describes how schematic sources are used in Foundation Series ISE projects. It also contains an overview of the basic concepts you need to use the ECS Schematic Editor and Symbol Editor tools. It contains the following sections:

- "Schematic Source Files"
- "Instantiating HDL Sources"
- "Simulating and Synthesizing Schematic Sources"
- "VHDL Functional Model"
- "Verilog Netlist"
- "ECS Schematic Editor"
- "VHDLgeneric Attribute Example"
- "Symbol Editor"
- "Symbol Libraries"
- "Guidelines for Creating Schematics"

Schematic Source Files

Your Foundation Series ISE project can include schematics as well as HDL sources to define your design. You initiate the creation of a schematic source from the Project Navigator. After the source file is created, the Project Navigator invokes the Engineering Capture System (ECS) tool for you to create and modify the schematic design. Only schematics created with the ECS tool can be used with Foundation Series ISE projects.

There are three principle components to ECS: the Schematic Editor, the Symbol Editor, and HDL netlisters. The Schematic Editor is the

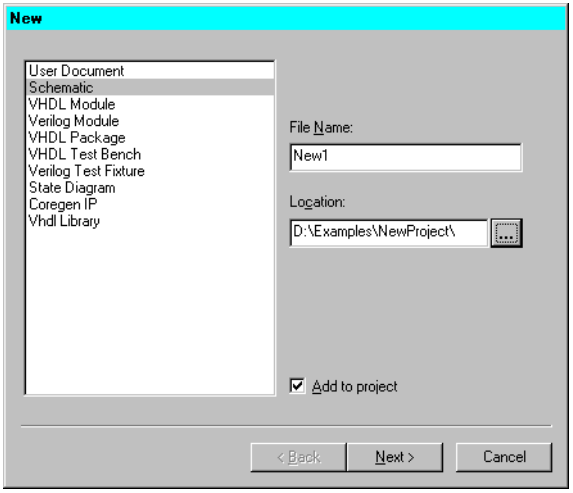
main schematic-creation interface. The Symbol Editor allows you to create and modify symbols that are used in the Schematic Editor. The netlister program translates a schematic into an HDL model that is used for synthesis and simulation of the design.

This section contains information on the Project Navigator's interaction with schematic source files. Refer to the "ECS Schematic Editor" section for information on the using the Schematic Editor and its interaction with the Symbol Editor. The most detailed information on using the ECS tools is in the Schematic Editor and Symbol Editor online help (from the **Help** menu and **F1** context sensitive help).

Creating a Schematic Source File

Use the following procedure to create and add a schematic source (.sch) to your project.

1. Open or create your project as described in the "Creating a Project" chapter.
2. Select **Project** → **New Source** from the Project Navigator menu to access the New source window.
3. Select **Schematic** in the New source window. An example New source window for an XST VHDL project is shown in the following figure.

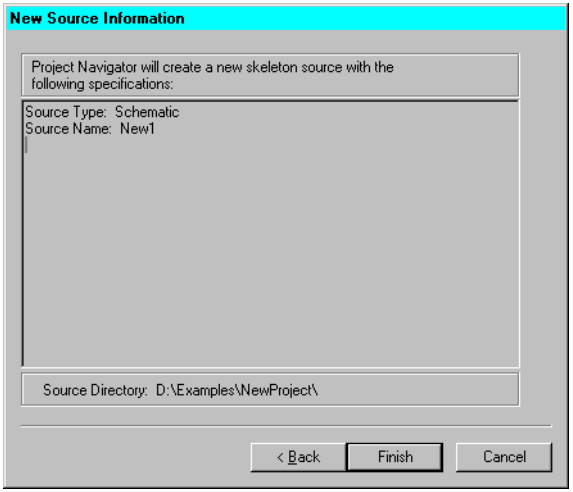


4. Enter a name for the new schematic in the File Name box. The project location is automatically entered in the Location box.

   Ensure that the Add to project box is checked if you want the schematic file added automatically to the project.

   Click **Next** when you are ready to proceed.

5. The New Source Information window appears.



   Click **Finish** to proceed.

6. The Project Navigator invokes the ECS Schematic Editor shown in the following figure. The Schematic Editor 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.

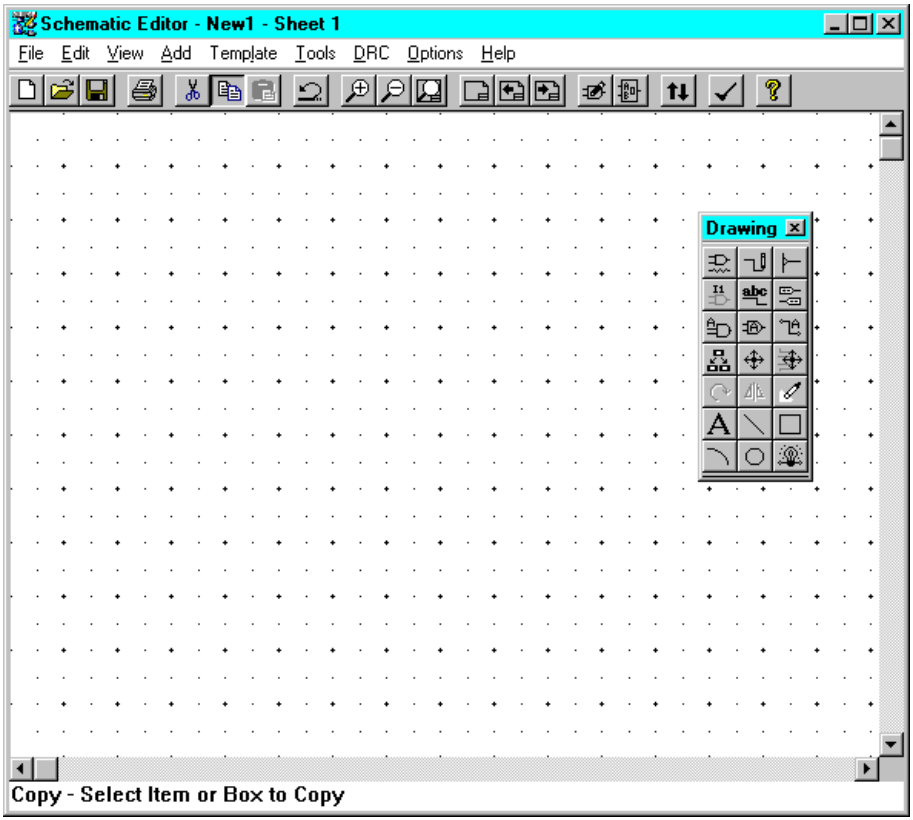
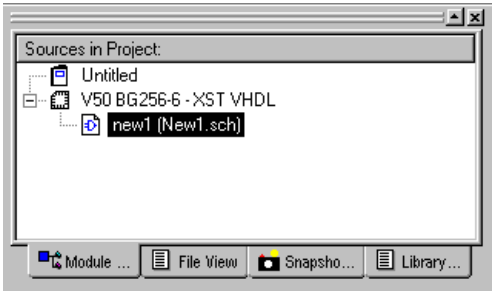


Figure 6-1 Schematic Editor Window with Drawing Toolbar

7. You can now use the Schematic Editor to enter the desired schematic module. Refer to the online help for details on using the Schematic Editor. Refer to the "Guidelines for Creating Schematics" section for important information on creating schematics for use with the synthesis tools.

If you selected to have this file added to the project, the new schematic file is automatically added to the project and shown in the Project Navigator Source window as shown in the following figure.



Opening a Schematic Source File

To open an existing schematic source (.sch) from the Project Navigator, double click on its name in the Source window. Or, select a schematic source in the Source window and then select **Source** → **Open** from the Project Navigator menu. The Schematic Editor 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). Whenever you modify symbols used in a your schematic source files, you need to run the **Update all Schematic Files** process to update all the schematics in your design with the current symbols in your symbol directories. You invoke this process by clicking on the Device / Synthesis Tool line in the Source window and then double clicking on **Update all Schematic Files** in the Process window.

The following types of errors when you open a schematic source indicate that you should run this process.

```
<program> ( schematicname.sch ) ERROR: Symbol SYMBOLNAME is Out of Date

<program> ( schematicname.sch ) ERROR: Check Library Paths or Update
Schematic

<program> ( schematicname.sch ) ERROR: Unable to Load Symbols for
schematicname.sch
```

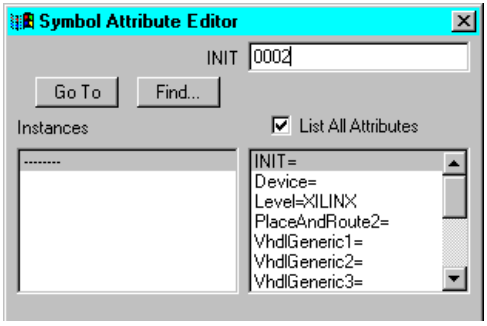
Xilinx Implementation Attributes/Constraints

It is possible to use the attribute function in ECS to pass constraint information from your schematic to the Xilinx implementation tools. Refer to Table 11-2 of the "Design Constraints / UCF File" chapter for a list of the implement constraints that can be used on ECS schematics. Refer to the ECS online help for detailed information on ECS schematic attributes.

Note Xilinx recommends that you enter the Xilinx implementation attributes that use the ECS PlaceAndRoute and PlaceAndRoute2 attributes (see Table 11-2 of the "Design Constraints / UCF File" chapter) only in the UCF and not on the ECS schematic.

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 as follows.

1. Create a new schematic.
2. Add the block RAM components.
3. Select **`Edit`** → **`Attribute`** → **`Symbol Attribute`** from the Schematic Editor menu.
4. Click on a block RAM symbol in the schematic.
5. Select the **`INIT=`** attribute from the Symbol Attribute Editor and enter the desired value in the input field as shown in the following figure. (You may need to click the List all Attributes check box to see the attribute list.)



6. Close the Symbol Attribute Editor dialog box. The INIT value is then 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, select an HDL source file in the Source window and the double click **`Create Schematic Symbol`** in the Process window. Symbols (.sym) created in this manner are automatically added to the Local symbol library and are available for use in the Schematic Editor.

Note Symbols are not allowed to have pin names beginning with the characters "B_" or "N_". If the HDL module contains pin names beginning with this sequence, an error is reported and the generated symbol is incomplete.

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. Use the following procedure to set this property.

1. Select an HDL source file in the Source window.
2. Right click on **`Create Schematic Symbol`** in the Process window.
3. Select **`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 within the Schematic Editor. From the Schematic Editor window, select **`View`** → **`Push/Pop`**. Then click on a symbol representing an instantiated HDL source in the 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 select **Create Schematic Symbol** in the Project Navigator Process 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

A typical way to use schematics in your Foundation Series ISE project is to create a top-level schematic and instantiate your HDL design sources into the schematic. The procedure to do this is as follows.

1. Create the HDL design sources and add them to the project.
2. Create a symbol for each HDL source by selecting that source in the Source window and clicking **Create Schematic Symbol** in the Process window. Symbols (.sym) created in this manner are automatically added to the Local symbol library used in the Schematic Editor.
3. Create a schematic source (.sch) for your project by selecting **Project** → **New Source** and picking "schematic" as the source type.
4. In the Schematic Editor, select **Symbol** → **Add**. Select each of the symbols created for your HDL sources from the Libraries / Directories window of the Symbol Libraries dialog box and place them on the schematic.
5. Select **Add** → **Wire** from the Schematic Editor menu and connect the HDL source symbols as appropriate. Save the schematic (.sch) and exit the Schematic Editor 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 Source window's design tree with the HDL sources listed under it.

Note The tutorial in the *Foundation Series 3.1i ISE Quick Start Guide* includes an example of using this procedure. The F1 online help for the Schematic Editor also includes detailed instructions on adding wires, naming nets and buses, adding I/O markers, etc. 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. Refer to 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 netlist for schematics. Refer to the “Verilog Netlist” section for more information on the generated netlist.

Simulation of schematic sources requires a testbench (VHDL) or test fixture (Verilog). You can use the HDL Bencher and the netlist generated from the schematic to create the testbench or test fixture. Refer to the “Creating a Testbench/Test Fixture” section of the “Simulation” chapter for information.

Refer to the “Simulation” chapter and the “Synthesis” chapter 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 set of 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.
- Busses 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 the ECS Schematic Editor, the Project Navigator uses the VHDL functional models for all further processing of the design.

You can view the VHDL functional model for a schematic by selecting the schematic in the Source window and then double-clicking on **`View VHDL Functional Model`** in the Process window. The VHDL model displays in the ISE Report Viewer. An example is shown in the following figure.

Note A separate error report (*schematic_name*.err) is generated for errors and messages concerning the functional model when it is opened in the Report Viewer. If the VHDL functional model file is maximized in the Report Viewer, the error report is not visible behind it. Minimize the VHDL functional model window (or reduce its size) to view the error report.

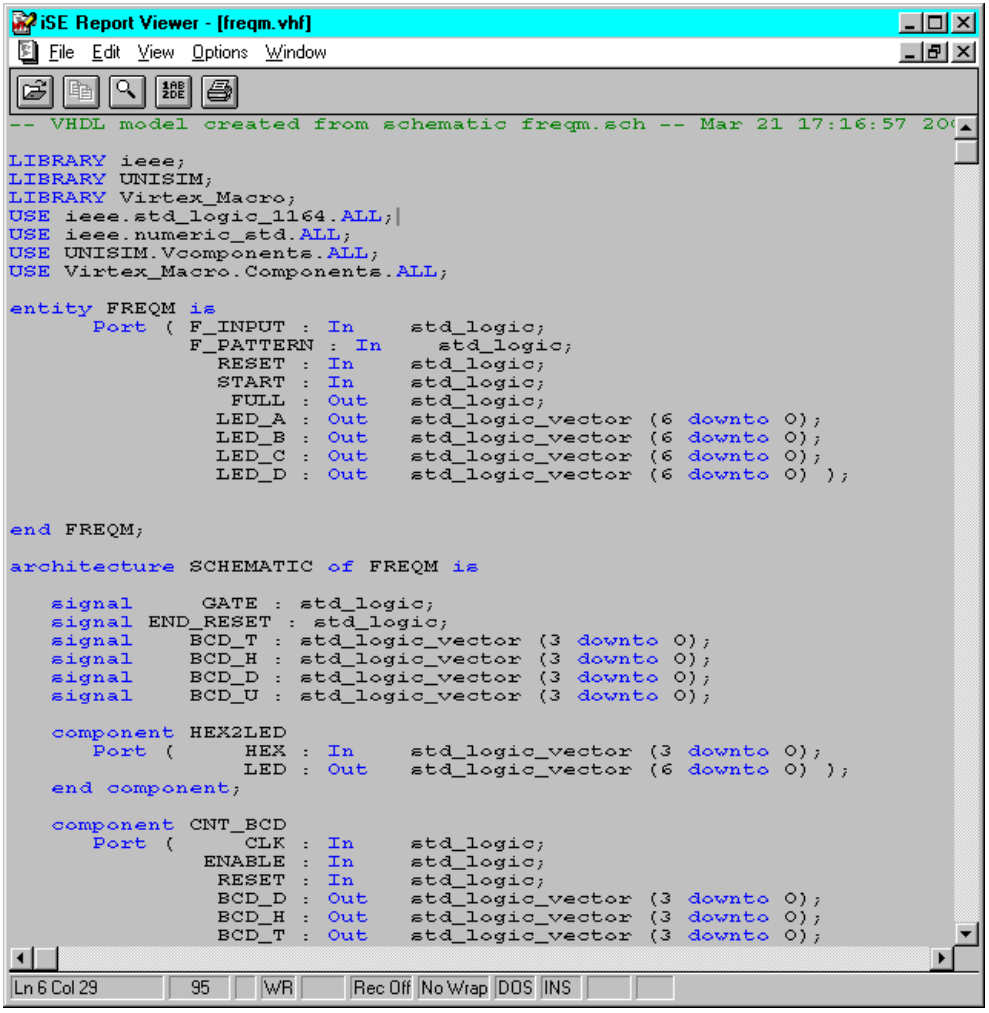


Figure 6-2 VHDL Functional Model Example

Setting VHDL Netlister Options

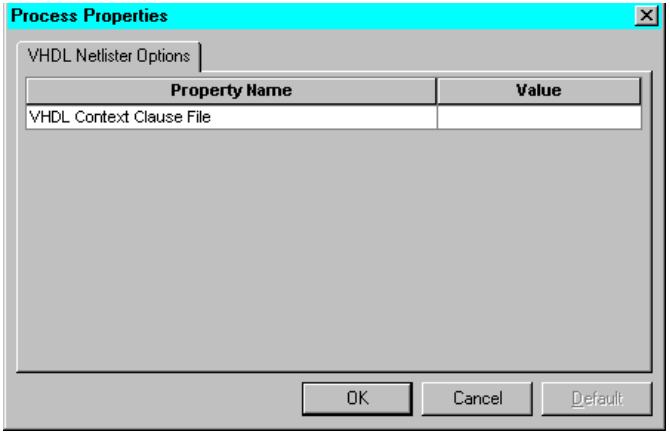
The VHDL model generated by the tools contains LIBRARY and USE statements based on the target architecture. For example:

```
[Generic]
LIBRARY ieee;
LIBRARY generics;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE generics.components.ALL;
```

You may specify additional libraries to be added to the VHDL model through a VHDL Context Clause File. This would be necessary, for example, if you have user-defined types, attributes, or component declarations in a package.

Use the following procedure to specify a VHDL Context Clause File to be used with VHDL netlist creation.

1. Select a schematic source (.sch) in the Project Navigator Source window.
2. Right-click on **`View VHDL Functional Model`** in the Process window.
3. Click **`Properties`**.
4. Enter the name and path of the VHDL Context Clause File in the Value field of the Process Properties dialog box (shown in the following figure. Click **`OK`**.



The following is an example of a VHDL Context Clause file:

```
[UTILITY]
LIBRARY utility;
USE utility.utility.all;
[SCH_FPGA]
LIBRARY sch_fpga;
USE sch_fpga.components.all;
[ALL_BDS]
LIBRARY l_hdl_bd;
USE l_hdl_bd.components.all;
LIBRARY l_mix_bd;
USE l_mix_bd.components.all;
LIBRARY l_sch_bd;
USE l_sch_bd.components.all;
```

The identifiers in brackets ([]) are section names. Following each section name should be a series of one or more LIBRARY and USE statements.

To select which LIBRARY and USE statements are added to a particular VHDL model, use the VHDLUse/Lib symbol attribute on the schematic's parent symbol. For example, assume that you needed the following context clauses in the VHDL model for a schematic named sample.sch:

```
LIBRARY ieee;
LIBRARY generics;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE generics.components.ALL;
LIBRARY utility;
USE utility.utility.all;
LIBRARY sch_fpga;
USE sch_fpga.components.all;
```

To get this set of context clauses, you would set the VHDLUse/Lib attribute for the symbol sample.sym to:

```
GENERIC,UTILITY,SCH_FPGA
```

Each section name listed in the attribute value will be searched for, starting with the Context Clause File named in this property, and if it not found there, then in vhdl.ini.

Verilog Netlist

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

You can view the Verilog netlist for a schematic by selecting the schematic in the Source window and then double-clicking on **`View Verilog Netlist`** in the Process window. The Verilog netlist displays in the ISE Report Viewer. An example is shown in the following figure.

Note A separate error report (*schematic_name*.err) is generated for errors and messages concerning the netlist when it is opened in the Report Viewer. If the Verilog Netlist file is maximized in the Report Viewer, the error report is not visible behind it. Minimize the Verilog netlist window (or reduce its size) to view the error report.

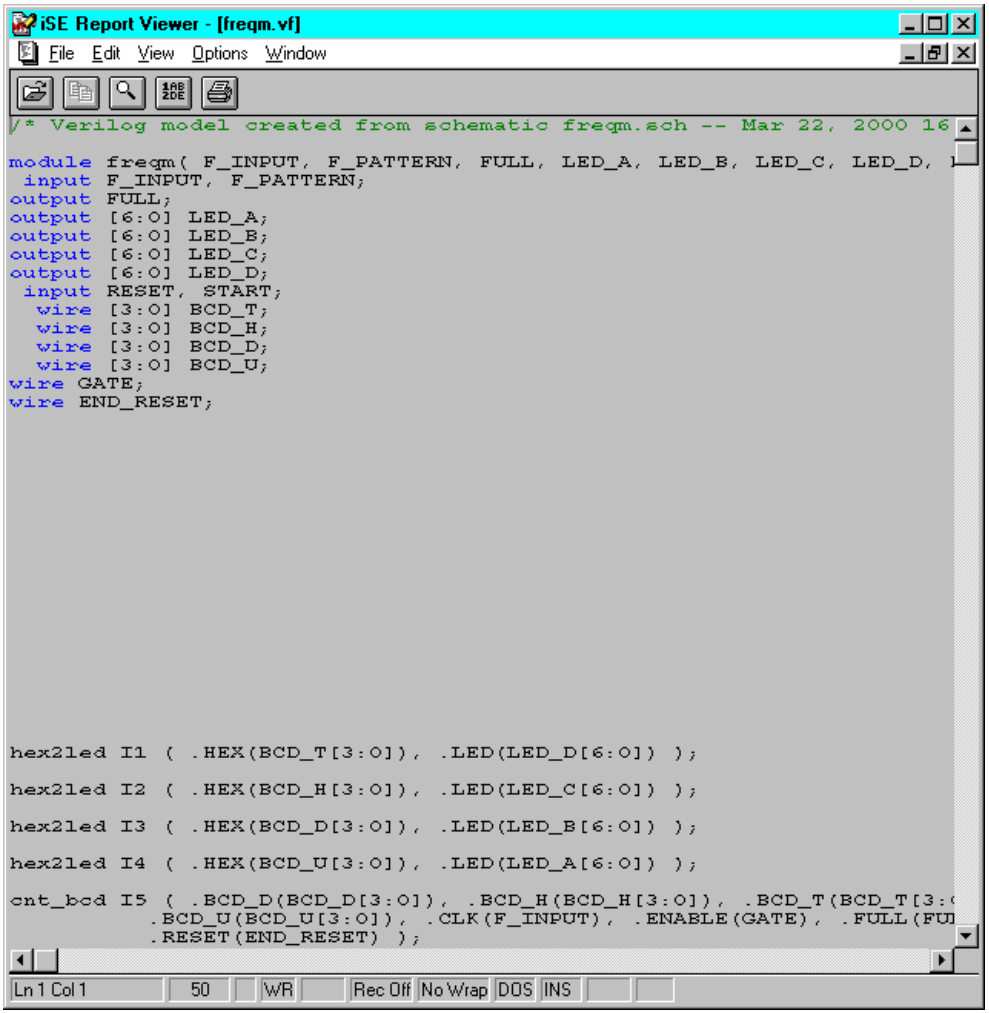


Figure 6-3 Verilog Netlist Example

ECS Schematic Editor

The ECS Schematic Editor is the design entry and analysis tool for schematic sources in Foundation Series ISE projects. It includes the following features:

- Schematic Capture

  The Schematic Editor captures your design logic in schematic form. The schematic file is continuously updated as you draw and is always ready for analysis.

- Libraries

  Xilinx device-specific symbol libraries are provided for schematic creation. You can create local symbols using the Symbol Editor as needed. Or, you can access ECS symbols from other projects.

- Symbol creation

  The Schematic Editor accesses the Symbol Editor where you can create your own symbols and give them whatever characteristics you want. Or, you can convert a schematic into a Block symbol to make your design easier to understand or for reuse in other projects.

- Consistency Check

  You can check your schematics at any time for such errors as unconnected wires or shorted nets.

- Electrical Check

  Electrical checks flag errors in connectivity and current loading on a completed design. It catches such errors as having too many loads connected to one output. This checking prevents electrical incompatibilities that would keep your design from working.

- Netlist Generation

  For use in Foundation Series ISE project, all schematics are converted into a VHDL or Verilog netlist depending on the synthesis tool selection for your project (see “Selecting a Device and Synthesis Tool” section of the “Creating a Project” chapter). When you select a schematic source in the Source window, you can select the **`View VHDL (or Verilog) Functional Model`** process in the Process window to see how the schematic was converted.

- Simulator Interface

  Testbenches and the ModelSIM simulators can be used to verify schematic designs. A testbench is required for simulation of the schematic's HDL functional model.

Note Detailed information on using the ECS Schematic Editor and Symbol Editor can be found in the ECS online help.

Schematic Editor Window

The Schematic Editor window is the main interface for the ECS tools. You access the Schematic Editor from the Project Navigator when you create a new schematic source for the project (see the "Creating a Schematic Source File" section) or when you double click on an existing schematic source file (.sch) in the Source window. The Schematic Editor window and an example design representing a schematic source named "freqm" are shown in the following figure.

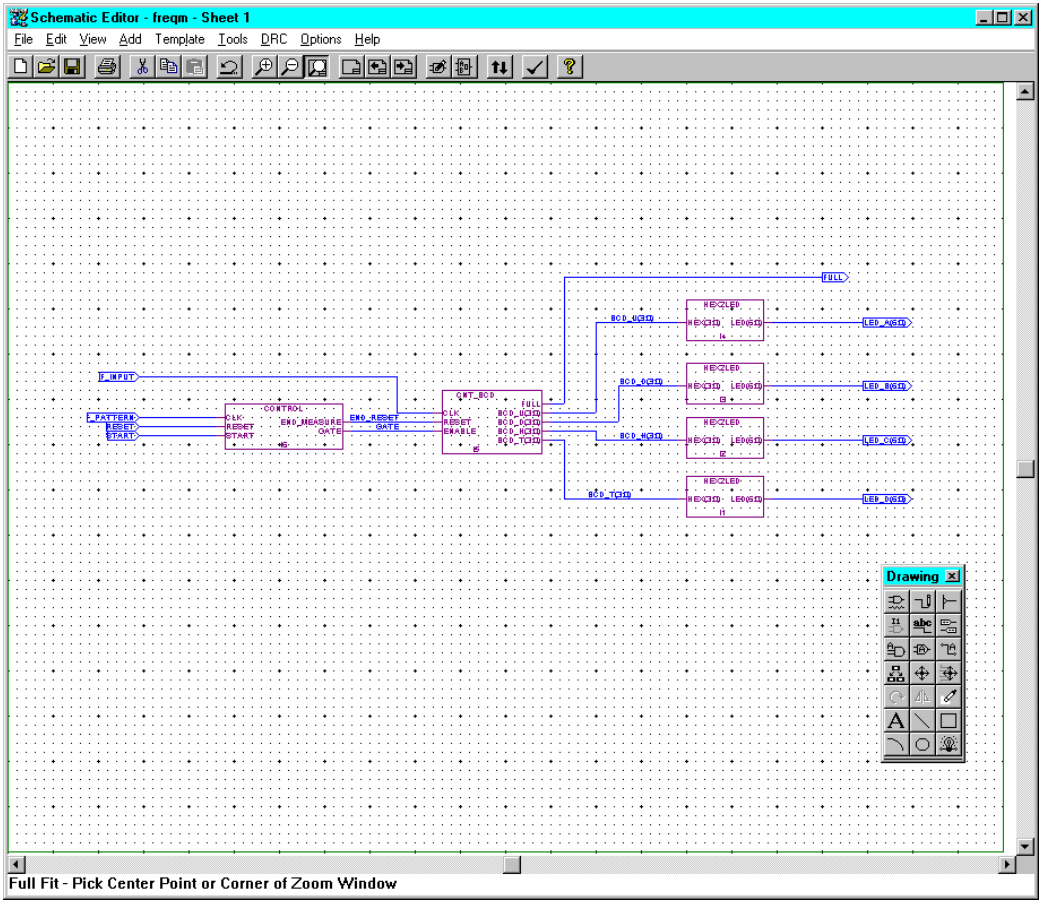


Figure 6-4 Schematic Editor Window with “freqm” Schematic

The top part of the window contains the *Title bar* with the name of the schematic and sheet number, the *Menu bar* with the main menu commands, and the *Toolbar* with icons for the commands. In Figure 6-4, the *Drawing Toolbar* is shown in the Schematic Editor window, but it can be positioned anywhere on your PC desktop.

The bottom section of the Schematic Editor window contains the *Prompt Line*. Messages and prompts in the Prompt Line (the very bottom line) guide you through the actions required to execute any command you select from the Schematic Editor Menu bar or Toolbar. You also enter text in the prompt line for such things as net names.

When minor errors occur (such as those that prevent a command from completing its action), the Schematic Editor replaces the scroll bar area above the Prompt Line with an *error window* that describes the error. Major errors are reported in pop-up message boxes.

The following list provides a functional overview for each command group on the Schematic Editor menu. Use the context sensitive online help command F1 to view more detailed help information for each main menu command group.

- File Menu

  Commands on the **File** menu create, open, and save schematics; create and manipulate schematic sheets; print schematic sheets and images; view design database statistics; and exit the Schematic Editor.

- Edit Menu

  You use the commands on the **Edit** menu to perform standard editing functions on schematic objects; move the location of symbol attributes; define, edit, and view attributes for pins, symbols and nets; edit tables; and launch the Symbol Editor.

- View Menu

  With commands on the **View** menu you can alter the viewing scale and location of the displayed schematic sheet within the Schematic Editor window. Commands include: Zoom In, Zoom Out, Full Fit, Pan, Push / Pop, and Redraw. You can also enable command shortcut toolbars from the View menu.

- Add Menu

  The **Add** menu commands are used to add wires, net names, bus taps, and I / O markers; add / modify symbols and symbol instances; assign / modify symbols, pins, and net attributes; add graphics to a schematic; and to add a table to a schematic.

- Template Menu

  You can use the Block Symbol command on the **Template** menu to generate a block symbol for hierarchical blocks without leaving the Schematic Editor. The Custom command is used to add a custom application program to the Template Menu.

- Tools Menu

  You can add custom application programs to the **Tools** Menu for use with your schematic designs.

- DRC Menu

  The Design Rules Check (**DRC**) menu contains commands to highlight a drawing object (e.g. pin, net, bus, or symbol); query the attributes of a drawing object; perform consistency and electrical checks; and add an application program to the DRC Menu.

- Options Menu

  Menus accessed from the **Options** menu contain items to customize the display of the items you place on the schematic sheet.

- Help Menu

  The **Help** Menu commands are used to access online help information about the product. The commands are used to view help contents, search for help on a topic, and view application specific help.

Action-Object Interface Examples

The Schematic Editor uses an *action-object* command structure. You must first select the *action* (usually from a menu or the drawing toolbar) then you select the *object* you want to act on.

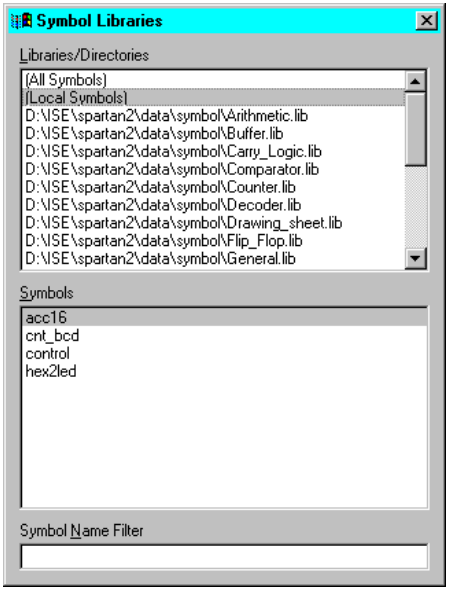
Note Be sure to check the Prompt Line at the bottom of the Schematic window for information on what action is required next as you create or modify your design.

The following sections provide examples of how the action-object interface works for some basic schematic entry tasks.

Adding a Symbol

Use the following action-object procedure to add an existing symbol to your schematic.

1. Select **Add** → **Symbol** from the Schematic Editor menu. Or, click on the Add Symbol icon in the drawing toolbar.
2. The Symbol Libraries dialog appears. An example is shown in the following figure.



3. Click on a library in the Libraries / Directories window to display its symbol list in the Symbol window. The local library contains the symbols you created in the Symbol Editor for the project. The Xilinx supplied libraries are divided into component categories. You can select **`All Symbols`** to view an alphabetical list of all available symbols; this includes the Xilinx-supplied libraries and the local library. Refer to the "Using Symbols from Other Projects" section for information on making more symbols available in the Symbol Library selection window.
4. Click on the desired symbol in the Symbol window and drag it to the schematic for placement.
5. Click to place the symbol.

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

As you place each symbol, the Schematic Editor automatically gives the symbol a unique *instance name* of the form I_*nn* (where *nn* is an integer). The instance name identifies the symbol to the Schematic Editor and netlister programs. You can change the instance name using **`Add`** → **`Instance Name`**. The Editor does not allow you to repeat an existing name.

The Schematic Editor lets you define an iterated instance in which a single symbol represents many instances of that symbol. An iterated instance is created simply by giving a symbol instance an instance name that includes a [*numberlist*]. For example, the instance name I1[7:0] would create eight instances of the corresponding symbol.

Adding a Wire

The action-object procedure to add a wire is as follows:

1. Select the action: **Add** → **Wire** from the Schematic Editor menu. Or, click on the Add Wire icon in the drawing toolbar.
2. Position the cursor in the schematic at the point you want to add the wire.
3. Click at the point where the wire should start.
4. Move the cursor to the point in the schematic and click at the desired end point.

You can continue to add wires or press **Esc** to exit the Add → Wire action.

Dragging a Wire

The action-object procedure to drag (move but not discount) a wire is as follows:

1. Select the action: **Edit** → **Drag** from the Schematic Editor menu. Or, click on the Drag icon in the drawing toolbar.
2. Position the cursor in the schematic on the wire you want to drag to a different position.
3. Click on the wire. The wire attaches to the cursor.
4. Drag the wire to its new location and click to reposition the wire.

You can continue to add wires or press **Esc** to exit the Edit → Drag action.

Note Dragging a wire repositions the wire without removing its connections. Moving a wire (**Edit** → **Move**) disconnects the wire when it is repositioned.

Removing a Symbol

Use the following action-object procedure to remove a symbol (or any item) from the schematic.

1. Select the action: **Edit** → **Remove** from the Schematic Editor menu.
2. Position the cursor in the schematic on the symbol you want to remove.
3. Click to remove the selected symbol from the schematic.

You can continue to remove items (symbols, wires, I/O markers, etc.) from the schematic or press **Esc** to exit the Edit → Remove action.

Panning, Zooming, Full Fit Operations

The action-object interface also applies to the basic schematic viewing/navigation processes such as panning, zooming, and fitting the schematic in the Schematic Editor window. For example, the action-object procedure to zoom in and pan in a schematic is as follows.

1. With a schematic open in the Schematic Editor window, select **View** → **Zoom In** from the Schematic Editor menu. The cursor becomes a "Z."
2. Position the Z cursor over the center point in the schematic where you want to zoom in.
3. Click to zoom in on the selected area. You can click as many times as necessary to get the desired size. (Selecting a different action or using **Esc** cancels the View → Zoom In action.)
4. Select **View** → **Pan** from the Schematic Editor menu.
5. Place the Z cursor in the schematic and click. The area where you clicked is moved to the center of the window.
6. You can move the cursor and click as many times as necessary to get the desired view. (Selecting a different action or using **Esc** cancels the View → Pan action.)

The Full Fit action works in the same manner as described above for zooming and panning. First select **View** → **Full Fit** then move the Z cursor over the schematic in the Schematic window and click.

Concepts Required to Use the Schematic Editor

A schematic created in the ECS Schematic Editor 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) with the Symbol Editor, 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, and attributes.

- *Graphics* are instructions that tell the Symbol Editor, Schematic Editor, and Hierarchy Navigator how to draw the symbol.
- *Text* labels the symbol, or adds supplemental information.
- *Pins* provide electrical connection between the symbol and the schematic's wiring.
- *Attributes* are parameters that describe the symbol's electrical behavior, the symbol's component parts (for example, its pins), and a number of other useful characteristics.

The following sections explain graphics, pins, and attributes in more detail

Graphics

Graphics are pictures of the symbols. 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.

Pins

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, then 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 There is only one kind of pin you can add (using **Add** → **Pin** in the Symbol Editor). 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]), or a range of numbers separated by a : (example: [8:15]), or both (example: [1,3,5,8:15]).

Attributes

Each symbol has a number of predefined attributes that describe its part number, component 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. This 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 in the Symbol Editor, it becomes the default value and is used with every instance of that symbol.

Refer to the "VHDLgeneric Attribute Example" section for an example of setting and using an ECS attribute.

ECS attributes can also be set on schematic components for implementation processing. Refer to the "Xilinx Implementation Attributes / 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.

There are two types of wires: scalar, which represent a single electrical connection, and buses, which represent multiple electrical connections.

Wires are added to schematics using **Add** → **`Wire`** or **Add** →**`Netname`** in the Schematic Editor.

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 an entire bus to

a bus pin on a symbol, assuming that the size of the bus and the size of the bus pin (i.e., 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 when you need to route 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.

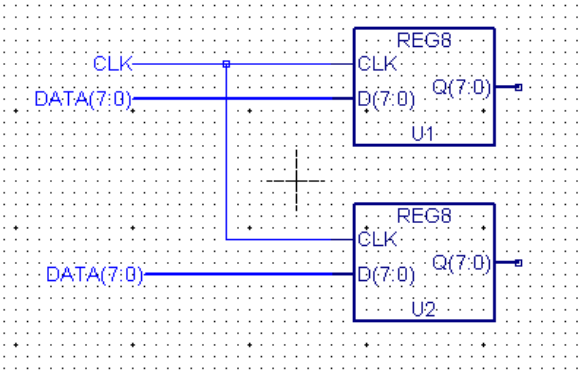
There is only one kind of wire you can add (using **Add** → **Wire**). Whether it is a net or a bus depends on how you name the wire (using the **Add** → **Netname**). Buses are named as *busname*[*numberlist*] where *busname* is the name of the bus and *numberlist* is a list of numbers separated by commas (example: [1,3,5]), or a range of numbers separated by a : (example: [8:15]), or both (example: [1,3,5,8:15]).

You can also draw and name a net or bus by selecting **Add** → **Netname** and dragging on the schematic to draw the wire.

Wires and Net Names

Wires are used to connect symbol pins on a schematic. Every wire must have a net name, which serves to identify the wire to the Schematic Editor and netlister programs. 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 will be electrically connected.

To illustrate this concept, consider the following 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 name 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. If you don't give a wire a name, the Schematic Editor automatically supplies one, of the form N_*n* (where *n* is an integer).

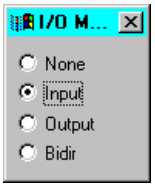
Net Attributes

Like symbols and symbol pins, nets (the wiring that connects symbols to each other and makes external connections) can also have attributes. Net attributes can describe characteristics associated with nets. Good examples are the KEEP and SAVE attributes.

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, then 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 the Schematic Editor. When you execute the **Add** → **IO** Marker command, the following dialog box appears:



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

- Clicking on the individual net name(s) 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, then 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 this context, "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. Or, 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 the

company logo and blank spaces for the project name, schematic sheet number, and so on.

The title block is a symbol. The title block template (the “tblock” symbol) is located in the General symbol library. Refer to 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.

Text is added to schematics using **Add** → **Text** in the Schematic Editor.

VHDLgeneric Attribute Example

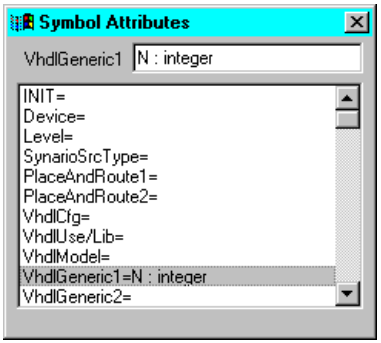
A common use of ECS attributes is to define generic parameter declarations for instantiated VHDL macros. A series of VHDLGeneric attributes are provided for this purpose. An example of using a VHDLGeneric attribute to assign a generic map statement to an instantiated VHDL macro on an ECS schematic is as follows.

Consider the following generic parameter declaration for entity FDC_18:

```
entity FDC_18 is
generic (N : integer := 18);
port (C,CLR : in std_logic;
      D : in std_logic_vector((N-1) downto 0);
      Q : out std_logic_vector((N-1) downto 0);
end FDC_18;
```

To set an attribute for the generic parameter declaration for FDC_18 symbols on the schematic, you would use the following procedure:

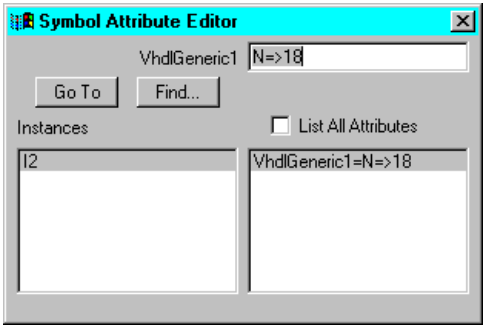
1. Select **Edit** → **Symbol** from the Schematic Editor.
2. Click on an FDC_18 instance in the schematic to access the Symbol Editor.
3. In the Symbol Editor, select **Edit** → **Attributes** → **Symbol Attributes**.
4. In the Symbol Attributes dialog box, set the **VhdlGeneric1** attribute to **N : Integer** as shown in the following figure.



5. Close the Symbol Attributes dialog box. Then close the Symbol Editor.

The VHDLGeneric attribute must also be defined on the symbol instance for those instances where you want to override the default value. In this case, you use the syntax that should appear in the generic map statement directly.

1. In the Schematic Editor, select **`Edit`** → **`Attributes`** → **`Symbol Attributes`**.
2. In the Symbol Attributes dialog box, set the **`VhdlGeneric1`** attribute to **`N=>18`** as shown in the following figure.



3. Close the Symbol Attributes dialog box.

The following VHDL functional model for a top-level schematic (test.sch) containing the FDC_18 entity illustrates the results of setting the VHDLGeneric1 attribute.

```
-- VHDL model created from schematic test.sch --Apr 07 12:32:12 2000

LIBRARY ieee;
LIBRARY UNISIM;
LIBRARY Virtex2_Macro;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE UNISIM.Vcomponents.ALL;
USE Virtex2_Macro.Components.ALL;

entity TEST is
    Port ( clk : In   std_logic;
           clr : In   std_logic;
           a   : In   std_logic_vector (17 downto 0);
           b   : In   std_logic_vector (17 downto 0);
           q   : Out  std_logic_vector (35 downto 0) )
end TEST;

architecture SCHEMATIC of TEST is
  signal a_temp : std_logic_vector (17 downto 0);
  signal c_temp : std_logic_vector (17 downto 0);
  signal q_temp : std_logic_vector (35 downto 0);

  component FDC_36
  Generic ( N : integer );
    Port ( C   : In   std_logic;
           CLR : In   std_logic;
           D   : In   std_logic_vector (35 downto 0);
           Q   : Out  std_logic_vector (35 downto 0) );
  end component;

  component FDC_18
  Generic ( N : integer );
    Port ( C   : In   std_logic;
           CLR : In   std_logic;
           D   : In   std_logic_vector (17 downto 0);
           Q   : Out  std_logic_vector (17 downto 0) );
  end component;

  attribute fpga_dont_touch : string;
  attribute fpga_dont_touch of I1 : label is "true";

begin
```

```
I4 : FDC_36
Generic Map ( N => 36               )
  Port Map (C=>clk,CLR=>clr,D(35)=>q_temp(35),D(34)=>q_temp(34),
    D(33)=>q_temp(33), D(32)=>q_temp(32), D(31)=>q_temp(31),
    D(30)=>q_temp(30), D(29)=>q_temp(29), D(28)=>q_temp(28),
    D(27)=>q_temp(27), D(26)=>q_temp(26), D(25)=>q_temp(25),
    D(24)=>q_temp(24), D(23)=>q_temp(23), D(22)=>q_temp(22),
    D(21)=>q_temp(21), D(20)=>q_temp(20), D(19)=>q_temp(19),
    D(18)=>q_temp(18), D(17)=>q_temp(17), D(16)=>q_temp(16),
    D(15)=>q_temp(15), D(14)=>q_temp(14), D(13)=>q_temp(13),
    D(12)=>q_temp(12), D(11)=>q_temp(11), D(10)=>q_temp(10),
    D(9)=>q_temp(9), D(8)=>q_temp(8), D(7)=>q_temp(7),
    D(6)=>q_temp(6), D(5)=>q_temp(5), D(4)=>q_temp(4),
    D(3)=>q_temp(3), D(2)=>q_temp(2), D(1)=>q_temp(1),
    D(0)=>q_temp(0), Q(35)=>q(35), Q(34)=>q(34),
    Q(33)=>q(33), Q(32)=>q(32), Q(31)=>q(31), Q(30)=>q(30),
    Q(29)=>q(29), Q(28)=>q(28), Q(27)=>q(27), Q(26)=>q(26),
    Q(25)=>q(25), Q(24)=>q(24), Q(23)=>q(23), Q(22)=>q(22),
    Q(21)=>q(21), Q(20)=>q(20), Q(19)=>q(19), Q(18)=>q(18),
    Q(17)=>q(17), Q(16)=>q(16), Q(15)=>q(15), Q(14)=>q(14),
    Q(13)=>q(13), Q(12)=>q(12), Q(11)=>q(11), Q(10)=>q(10),
    Q(9)=>q(9), Q(8)=>q(8), Q(7)=>q(7), Q(6)=>q(6),
    Q(5)=>q(5), Q(4)=>q(4), Q(3)=>q(3), Q(2)=>q(2),
    Q(1)=>q(1), Q(0)=>q(0) );
I2 : FDC_18
Generic Map ( N=>18                 )
  Port Map ( C=>clk, CLR=>clr, D(17)=>a(17), D(16)=>a(16),
    D(15)=>a(15), D(14)=>a(14), D(13)=>a(13), D(12)=>a(12),
    D(11)=>a(11), D(10)=>a(10), D(9)=>a(9), D(8)=>a(8),
    D(7)=>a(7), D(6)=>a(6), D(5)=>a(5), D(4)=>a(4),
    D(3)=>a(3), D(2)=>a(2), D(1)=>a(1), D(0)=>a(0),
    Q(17)=>a_temp(17), Q(16)=>a_temp(16), Q(15)=>a_temp(15),
    Q(14)=>a_temp(14), Q(13)=>a_temp(13), Q(12)=>a_temp(12),
    Q(11)=>a_temp(11), Q(10)=>a_temp(10), Q(9)=>a_temp(9),
    Q(8)=>a_temp(8), Q(7)=>a_temp(7), Q(6)=>a_temp(6),
    Q(5)=>a_temp(5), Q(4)=>a_temp(4), Q(3)=>a_temp(3),
    Q(2)=>a_temp(2), Q(1)=>a_temp(1), Q(0)=>a_temp(0) );
I3 : FDC_18
Generic Map ( N => 18               )
  Port Map ( C=>clk, CLR=>clr, D(17)=>b(17), D(16)=>b(16),
    D(15)=>b(15), D(14)=>b(14), D(13)=>b(13), D(12)=>b(12),
    D(11)=>b(11), D(10)=>b(10), D(9)=>b(9), D(8)=>b(8),
    D(7)=>b(7), D(6)=>b(6), D(5)=>b(5), D(4)=>b(4),
    D(3)=>b(3), D(2)=>b(2), D(1)=>b(1), D(0)=>b(0),
```

```
      Q(17)=>c_temp(17), Q(16)=>c_temp(16), Q(15)=>c_temp(15),
      Q(14)=>c_temp(14), Q(13)=>c_temp(13), Q(12)=>c_temp(12),
      Q(11)=>c_temp(11), Q(10)=>c_temp(10), Q(9)=>c_temp(9),
      Q(8)=>c_temp(8), Q(7)=>c_temp(7), Q(6)=>c_temp(6),
      Q(5)=>c_temp(5), Q(4)=>c_temp(4), Q(3)=>c_temp(3),
      Q(2)=>c_temp(2), Q(1)=>c_temp(1), Q(0)=>c_temp(0) );
  I1 : MULT18X18
    Port Map (A(17)=>a_temp(17),A(16)=>a_temp(16), A(15)=>a_temp(15),
      A(14)=>a_temp(14), A(13)=>a_temp(13), A(12)=>a_temp(12),
      A(11)=>a_temp(11), A(10)=>a_temp(10), A(9)=>a_temp(9),
      A(8)=>a_temp(8), A(7)=>a_temp(7), A(6)=>a_temp(6),
      A(5)=>a_temp(5), A(4)=>a_temp(4), A(3)=>a_temp(3),
      A(2)=>a_temp(2), A(1)=>a_temp(1), A(0)=>a_temp(0),
      B(17)=>c_temp(17), B(16)=>c_temp(16), B(15)=>c_temp(15),
      B(14)=>c_temp(14), B(13)=>c_temp(13), B(12)=>c_temp(12),
      B(11)=>c_temp(11), B(10)=>c_temp(10), B(9)=>c_temp(9),
      B(8)=>c_temp(8), B(7)=>c_temp(7), B(6)=>c_temp(6),
      B(5)=>c_temp(5), B(4)=>c_temp(4), B(3)=>c_temp(3),
      B(2)=>c_temp(2), B(1)=>c_temp(1), B(0)=>c_temp(0),
      P(35)=>q_temp(35), P(34)=>q_temp(34), P(33)=>q_temp(33),
      P(32)=>q_temp(32), P(31)=>q_temp(31), P(30)=>q_temp(30),
      P(29)=>q_temp(29), P(28)=>q_temp(28), P(27)=>q_temp(27),
      P(26)=>q_temp(26), P(25)=>q_temp(25), P(24)=>q_temp(24),
      P(23)=>q_temp(23), P(22)=>q_temp(22), P(21)=>q_temp(21),
      P(20)=>q_temp(20), P(19)=>q_temp(19), P(18)=>q_temp(18),
      P(17)=>q_temp(17), P(16)=>q_temp(16), P(15)=>q_temp(15),
      P(14)=>q_temp(14), P(13)=>q_temp(13), P(12)=>q_temp(12),
      P(11)=>q_temp(11), P(10)=>q_temp(10), P(9)=>q_temp(9),
      P(8)=>q_temp(8), P(7)=>q_temp(7), P(6)=>q_temp(6),
      P(5)=>q_temp(5), P(4)=>q_temp(4), P(3)=>q_temp(3),
      P(2)=>q_temp(2), P(1)=>q_temp(1), P(0)=>q_temp(0) );

end SCHEMATIC;
```

Symbol Editor

The Symbol Editor is used to create and edit symbols and symbol attributes. The Symbol Editor is accessed only from the Schematic Editor. Therefore, many of the Symbol Editor operations are initiated in the Schematic Editor.

Symbol Editor Window

You can open the Symbol Editor from the Schematic Editor by selecting **`Edit`** → **`Symbol`** and then either entering the name of the symbol you want to edit in the Prompt Line or clicking on the symbol in the schematic. To open the Symbol Editor to create a new block symbol, select **`Template`** → **`Block Symbol`** from the Schematic Editor menu. An example of the ECS Symbol Editor window is shown in Figure 6-5.

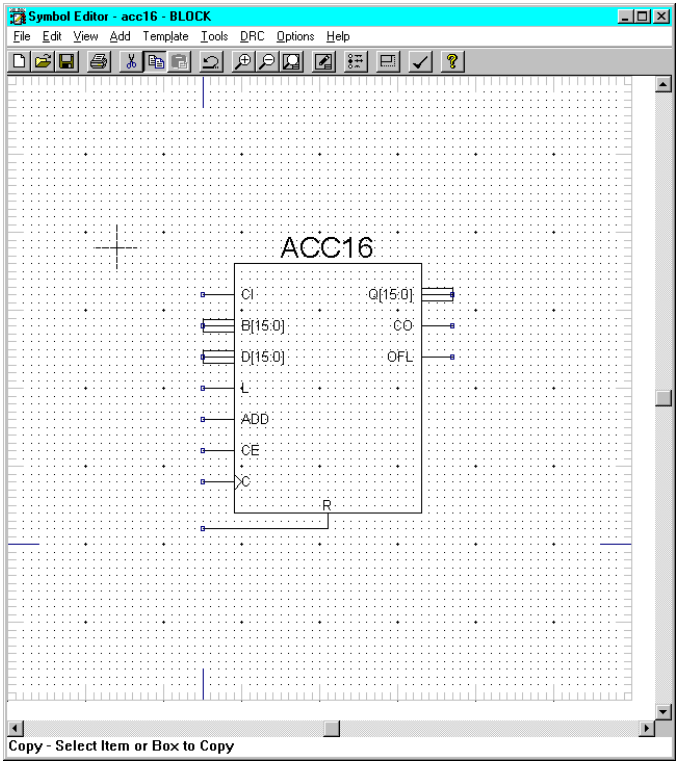


Figure 6-5 ECS Symbol Editor

The following list provides a functional overview for each main menu command group. Use the context sensitive online help command F1 to view more detailed help information for each main menu command group.

- File Menu

  Used to create, open, and save symbols; print symbol sheets and images; and exit the Symbol Editor.

- Edit Menu

  Used to perform standard editing functions on symbol objects; define, edit, and view symbol and pin attributes; move the location of pin attributes; set the symbol origin; expand the symbol page size; change the symbol type; and launch the Schematic Editor.

- View Menu

  Used to alter the viewing scale and location within the Symbol Editor window (Zoom In, Zoom Out, Full Fit, Pan, and Redraw commands); and to enable command shortcut toolbars.

- Add Menu

  Used to add pins and graphics to a symbol.

- Template Menu

  Used to add an application program to the Template Menu.

- Tools Menu

  Used to add application programs to the Tool Menu.

- DRC Menu

  Used to perform a design consistency check and add an application program to the DRC Menu.

- Options Menu

  Used to set design configuration preferences.

- Help Menu

  The Help Menu commands are used to access online help information about the product. The commands are used to view help contents, search for help on, and view domain specific help.

Symbol Types

The Symbol Editor creates three different kinds of symbols: block symbols, master symbols, and graphic 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. They 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 so on.

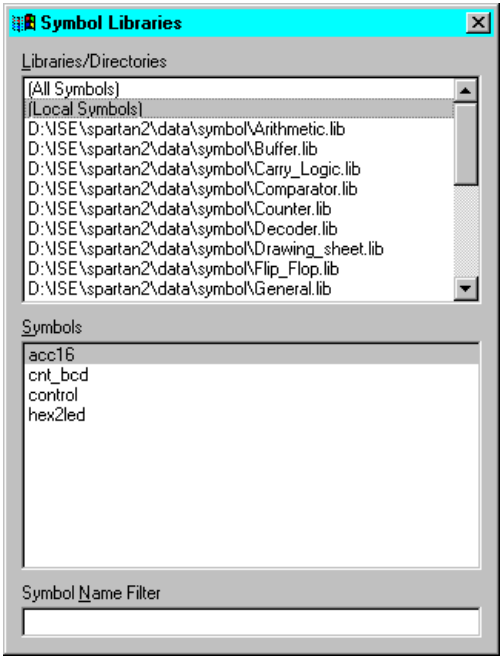
Master symbols cannot be freely placed on a schematic sheet. Instead, they are automatically positioned at one of the corners. This permits resizing the sheet without having to move the title block (or other annotations). The sheet corner is determined by the location of the symbol's origin. If the origin is placed at the upper-right corner of the Master symbol (for example), the symbol will be positioned at the upper-right corner of the sheet.

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

Symbol Libraries

The Schematic Editor automatically includes the symbol libraries specific to the Xilinx device targeted in your project. Symbols that you created in your project are added to the project's Local symbol library. You an 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 Libraries dialog box accessed from **Add** → **Symbol** in the Schematic Editor. The following figure shows an example Symbol Library selection window.



Modifying an Existing Symbol

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

Use the following procedure to modify a symbol in the Local directory.

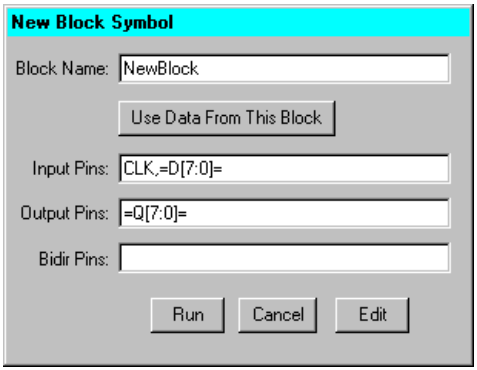
1. Select **Symbol** → **Edit** from the Schematic Editor menu.
2. You can do either of the following to specify the symbol to be edited:
   - If the symbol has been placed on the schematic, click on the symbol to be modified.
   - If the symbol is not currently on the schematic, enter its name in the Prompt Line.
3. The Symbol Editor opens with the selected symbol. Edit the symbol as desired.

   Note If you selected a symbol from the read-only Xilinx-supplied libraries, a dialog box appears before the Symbol Editor opens to allow you to confirm copying the symbol to the Local library for modification.
4. Save the symbol and exit the Symbol Editor. If the symbol was already on the schematic, it is immediately updated to reflect the changes. If it was not on the schematic, you can select the modified symbol from the Local library for use in your schematic.

Creating a New Block Symbol

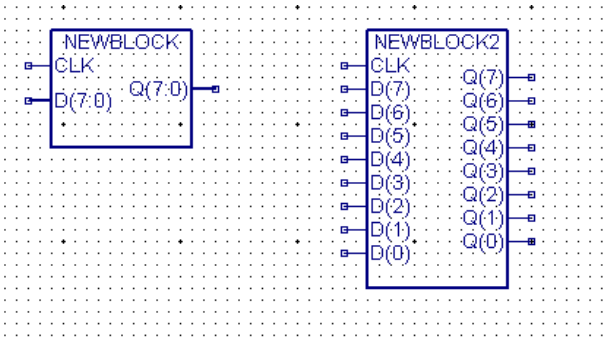
A template is available in the Schematic Editor for block symbol creation using the following procedure.

1. Select **Template** → **Block Symbol** from the Schematic Editor. The
2. Enter a Block Name and Pin names (separated by commas). You can use bus notation ([7:0], for example) when entering a pin name. To have a single bus pin created, you need to use equal signs (=) around the input pin name as shown in the following figure.



3. Click **Run** to create the new symbol.

   For bus pin names enclosed with equal signs (=), a single bus pin is generated as illustrated by the NEWBLOCK symbol in the following figure. For bus pin names not enclosed with equal signs, the bus is expanded into individual pins as illustrated in the NEWBLOCK2 symbol.



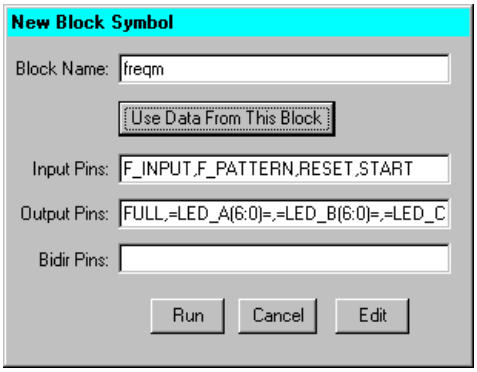
4. The symbol is created and attached to the cursor ready for placement. The symbol is automatically added to the Local library.

Creating a Block Symbol from a Schematic

Use the following procedure to create a new block symbol from an existing schematic.

1. Select **Template** → **Block Symbol** from the Schematic Editor menu.
2. Click **Use Data From this Block**.

As shown in the following figure. The schematic name is automatically placed in the Block Name field. The pins created in the schematic are automatically entered in the appropriate Pins fields.



3. Click **Run** 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, select the HDL module in the Source window of the Project Navigator and then double-click the **`Create Schematic Symbol`** process in the Process window. 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 Libraries selection window. You can include the project's symbols in the Symbol Libraries selection window on a device basis or on an all projects basis. When added on a device basis, the symbols are available only for projects that target a specified device. When added on an all-projects basis, the symbols become available to all projects regardless of the device targeted. In both cases, only symbols created in the ECS Schematic Editor are available for use in other projects.

Note The procedure to add symbol libraries uses the "inieditor." Use of this tool is currently supported only on a limited basis. For more

information on using the inieditor, please see Solution Record #8533 (`http://support.xilinx.com/techdocs/8533.htm`).

Device Basis

Use the following procedure to add the local symbols created in an ISE project to the Symbol Libraries selection window for all projects that target a specified device family.

1. Open an MSDOS command window (**`Start`** → **`Programs`** → **`Command Prompt`**).
2. Type **`iniedit $XILINX\data\`*`device_family`*`.ini`**

   where

   $XILINX is the path to where ISE is installed.

   device_family can be any of the following device families.

   `spartan`

   `spartan2`

   `spartanXL`

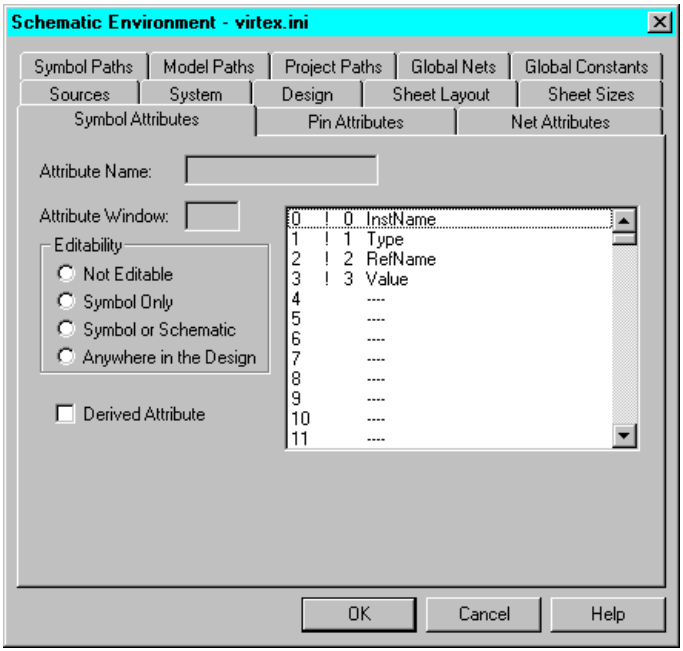
   `virtex`

   `virtex2`

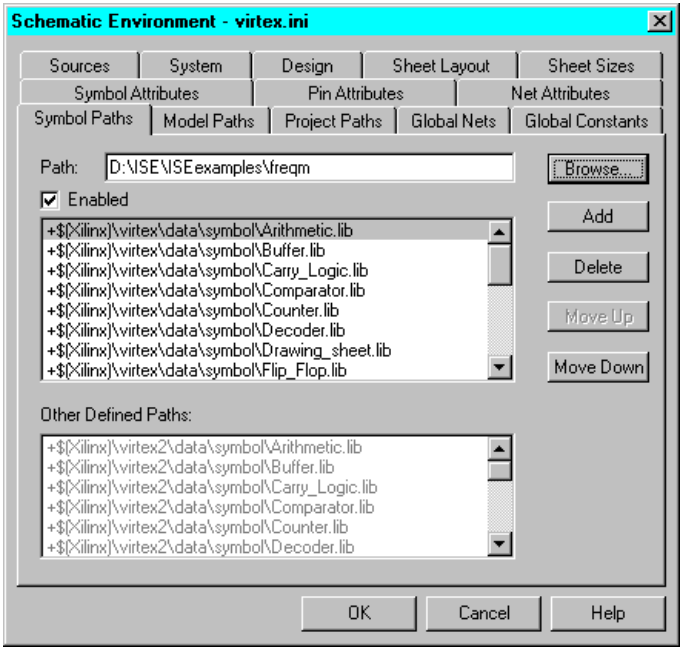
   `virtexe`

   `xc4000`

   `xc9000`
3. Press **`Enter`** to access a schematic environment dialog box for the specified device. For example, if you typed **iniedit D:\ISE\data\virtex.ini**, the following dialog box appears. If you typed **`spartan.ini`** instead of **`virtex.ini`**, the spartan dialog box would appear.

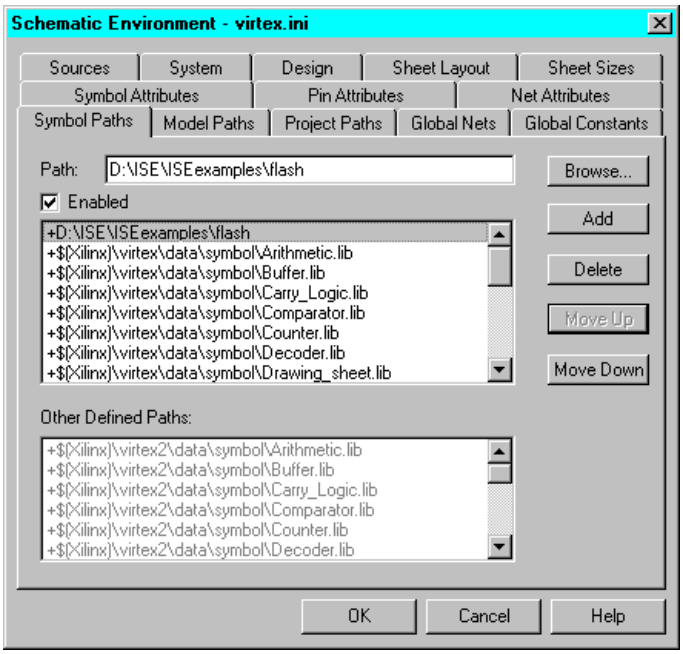


4. Click the Symbol Paths tab. The Symbol Paths tab for virtex.ini is shown in the following figure. (The path to the currently open project “freqm” appears in the Path field.)



5. Enter the path to the project whose Local symbols are be added to the Symbol Library selection window in the Path field. Or, click **Browse** 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. In the Browse dialog, change the Files of Type field to **Symbol Dirs (*.sym)** files. Click on any .sym file for the selected project and then click **Open** to return to the Schematic Environment dialog box.

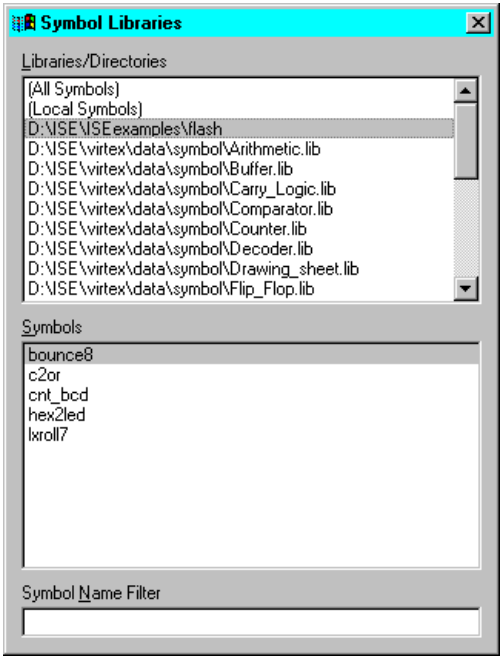
   The full path to the selected project (for example, the “flash” project) appears in the Path field of the Schematic Environment dialog box as shown in the following figure.



6. Click **Add** and then **OK** on the Symbol Paths tab to add the local symbols for the selected project (specified in the Path field) to the search path for all projects that use the specified device (for example, Virtex).

7. The symbols in the selected project are now available in the Schematic Editor's Symbol Library selection box (accessed by **Add** → **Symbol**) for all projects that use the selected device.

   For example, all projects targeting a Virtex device would now have access to the symbols created in the "flash" project as shown in the following figure.

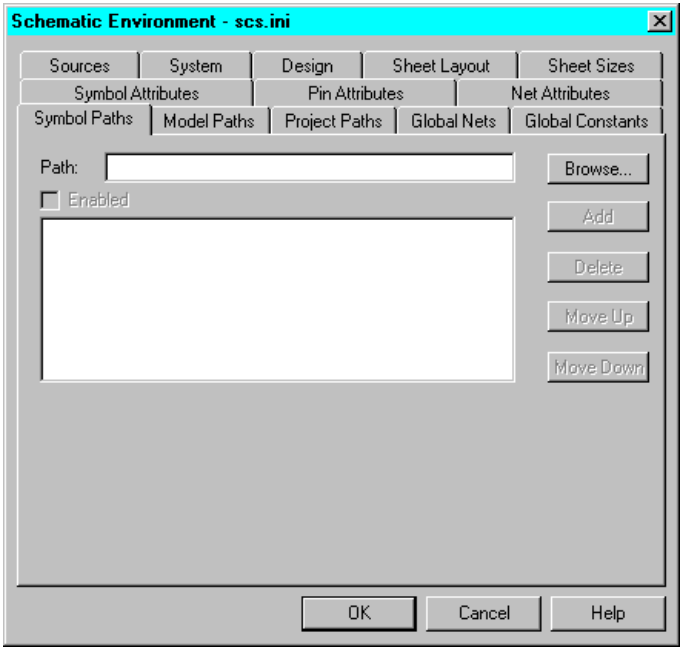


Note When you use symbols from other projects, be sure to edit the symbol and use the "Save As" command to place it in the Local library of the current project.

All Projects Basis

Use the following procedure to add the local symbols created in an ISE project to the Symbol Libraries selection window for all projects regardless of the targeted device for the project.

1. Open an MSDOS command window (**`Start`** → **`Programs`** → **`Command Prompt`**).
2. Type **`iniedit -master`**
3. Press **`Enter`** to access the master Schematic Environment (scs.ini) dialog box for the Schematic Editor.
4. Click the **`Symbol Paths`** tab (shown in the following figure).



5. Enter the path to the desired project in the Path field. Or, click **Browse** 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. Change the Files of Type field to **Symbol Dirs (*.sym)** files. In the Browse dialog box, click on any .sym file for the selected project and then click **Open** to return to the Schematic Environment dialog box.
6. Click **Add** and then **OK** on the Symbol Paths tab to add local symbols for the selected project (specified in the Path field) to the Symbol Libraries for all projects.
7. The symbols in the selected project are now available in the Schematic Editor's Symbol Library selection box (accessed by **Add** → **Symbol**) for all projects regardless of the targeted 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 and then processed by either the XST or FPGA Express synthesis engine. Because of this, the following guidelines must be followed when creating schematic sources.

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

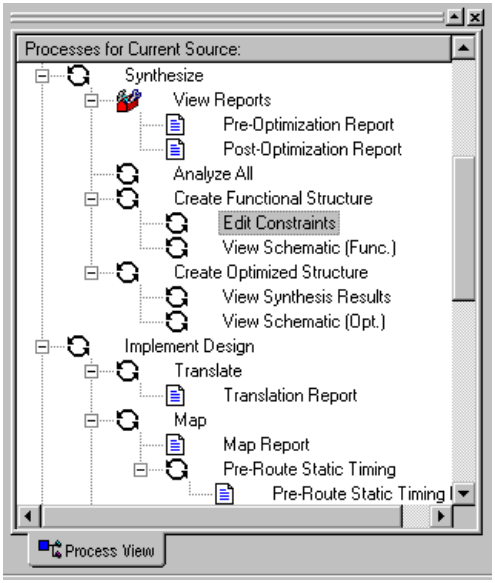
  Use I/O primitives such as IBUF, OBUF, IFD, IFDX, OFD, OFDX, etc. only. Do not use I/O macros such as IBUF4, IFD_1, etc. If I/O macros are used, the synthesis engine inserts an additional IBUF or OBUF at the port causing errors in the MAP process.
- (For VHDL Only) All input pins of the Xilinx unified 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 will be 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 will result in errors in MAP due to the clock pin being left unconnected.

  To access the Express Constraints Editor, select Edit Constraints (shown in the following figure) under Synthesis in the Process window.


Processes for Current Source:
Synthesize
View Reports
Pre-Optimization Report
Post-Optimization Report
Analyze All
Create Functional Structure
Edit Constraints
View Schematic (Func.)
Create Optimized Structure
View Synthesis Results
View Schematic (Opt.)
Implement Design
Translate
Translation Report
Map
Map Report
Pre-Route Static Timing
Pre-Route Static Timing
Process View

Chapter 7

State Diagrams

For state machine design entry, Foundation Series ISE includes integrated support for StateCAD® and StateBench™ from Visual Software Solutions, Inc. (VSS). This chapter describes the integration of the supported state diagram entry tools with Foundation Series ISE. It contains the following sections:

- "StateCAD / StateBench"
- "Acquiring StateCAD / StateBench Tools"
- "Launching StateCAD"
- "Using StateBench"
- "Adding State Diagram Sources to Your Project"
- "Instantiating State Diagram Modules in HDL Designs"
- "Instantiating State Diagram Modules in Schematic Designs"
- "Using Foundation Series 2.1i State Diagrams"

StateCAD/StateBench

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, and an Optimization Wizard to maximize performance for the target device. StateCAD also identifies many kinds of design logic 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 Foundation Series ISE project.

The StateBench tool automatically creates VHDL test benches and Verilog test fixtures from StateCAD designs. You can use these to

verify the behavior of the StateCAD design. You can also add them to your project and use them as testbenches.

The basic procedure for creating and using state diagrams with Foundation Series ISE projects is as follows:

1. Create the state diagram as a new source for your project.
2. Create / modify the state diagram in StateCAD.
3. Verify the state diagram in StateCAD. Use StateBench, if desired.
4. StateCAD creates a synthesizable HDL file from the verified state diagram.
5. Add state diagram source files to your project. Add the state diagram (.dia file) to the project as a user document. Add the StateCAD-generated HDL file (.vhd, .v, or .abl) to the project as a source module / entity.

Acquiring StateCAD/StateBench Tools

The Foundation Series ISE 3.1i installation does not automatically install the StateCAD and StateBench tools. You can acquire and / or access these tools as described in the following sections.

Xilinx Edition

You can install a limited version of StateCAD / StateBench, called the Xilinx Edition, from the ALLSTAR CD included in your Foundation Series ISE 3.1i software package. The Xilinx Edition is provided free of charge and requires no licensing.

The Xilinx Edition of StateCAD is limited to 6 states, 16 transitions, and 2 equations per state diagram. If you register with VSS when you install the Xilinx Edition, the limits for the Xilinx Edition of StateCAD are increased to 10 states, 20 transitions, and 8 equations per state diagram. (An upgrade path is available from the Xilinx Edition to the complete edition.)

Use the following procedure to install the Xilinx Edition of StateCAD / StateBench:

1. Insert your ALLSTAR CD into your PC's CD-ROM drive.
2. In your Windows Start menu, select, **`Start`** → **`Programs`** → **`Xilinx Foundation Series ISE 3.1i`** → **`Partner`**

Products → **Install StateCAD Xilinx Edition (CD Required)**.

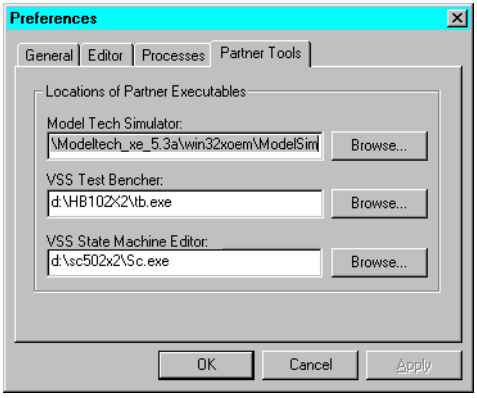
3. Follow the directions in the installation screens.

Pre-Existing or Upgraded StateCAD Tools

You do not need to install the Xilinx limited edition of StateCAD / StateBench to add State Machine Editor tools to Foundation Series ISE. If you currently have a fully functional version of StateCAD / StateBench or if you obtain it from VSS later, you can instruct Foundation Series ISE to use those tools with your project.

Use the following procedure to identify the StateCAD / StateBench installation you want to use with Foundation Series ISE:

1. From the Project Navigator Menu bar, select **Edit** → **Preferences**.
2. In the Preferences window, select the Partner Tools tab.



3. In the **VSS State Machine Editor** field on the Partner Tools tab, enter the path to the program executable for an existing installation of StateCAD / StateBench. Or, click **Browse** to select the path.

Complete Edition

You can obtain the unlimited, fully-functional version of StateCAD / StateBench from VSS. You can go the StateCAD website at `http://www.statecad.com` or product and licensing information.

All Foundation Series ISE users qualify for a $500 discount on the purchase of the StateCAD Complete Edition. To receive the discount, you need to provide your Host ID, which is located in StateCAD's **`Help`** → **`About`** dialog box. To purchase the StateCAD Complete Edition, contact Visual Software Solutions at (954) 370-9030 or e-mail sales@statecad.com.

Sales and Support of StateCAD and StateBench

All sales, support, and licensing of StateCAD and StateBench are done through Visual Software Solutions, Inc. This includes support for the Xilinx Edition.

If you have questions regarding your StateCAD / StateBench software installation, contact VSS using the phone numbers or e-mail addresses that are shown during the installation process. You can also go to the product's website: `http://www.statecad.com`.

For the latest sales and support information, select **`Help`** → **`Support/Sales Info`** from on the StateCAD menu. Or, click the Sales / Support icon (shown below) on the StateCAD toolbar.

Launching StateCAD

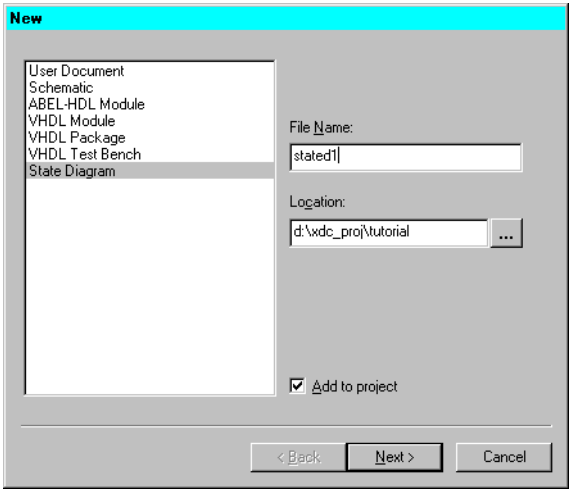
After you install StateCAD as described in the "Acquiring StateCAD / StateBench Tools" section, you can launch it from within the Project Navigator.

Note The procedures described in this section are the same for both the Xilinx Edition and the complete edition of StateCAD / StateBench, assuming the Project Navigator has been directed to the installed edition.

Creating a New State Diagram

To create a new state diagram source for use with your project, use the following procedure:

1. Select **`Project`** → **`New Source`** from the Project Navigator to display the New source dialog box.
2. Click **`State Diagram`** to select it from the list of source displayed in the New source dialog box.



3. Enter a name for the state diagram in the File Name field.

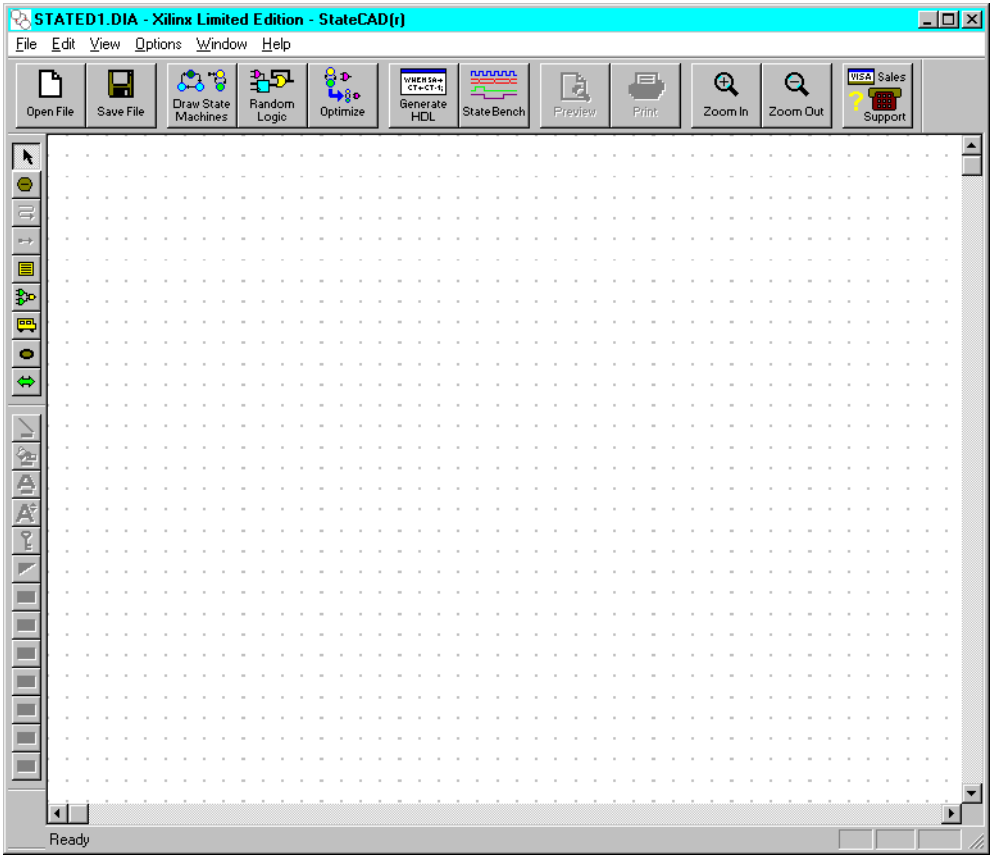
   A StateCAD state diagram name has the following requirements:

   - The name must begin with an alphabetic character only.
   - StateCAD is not case sensitive. Filenames 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.

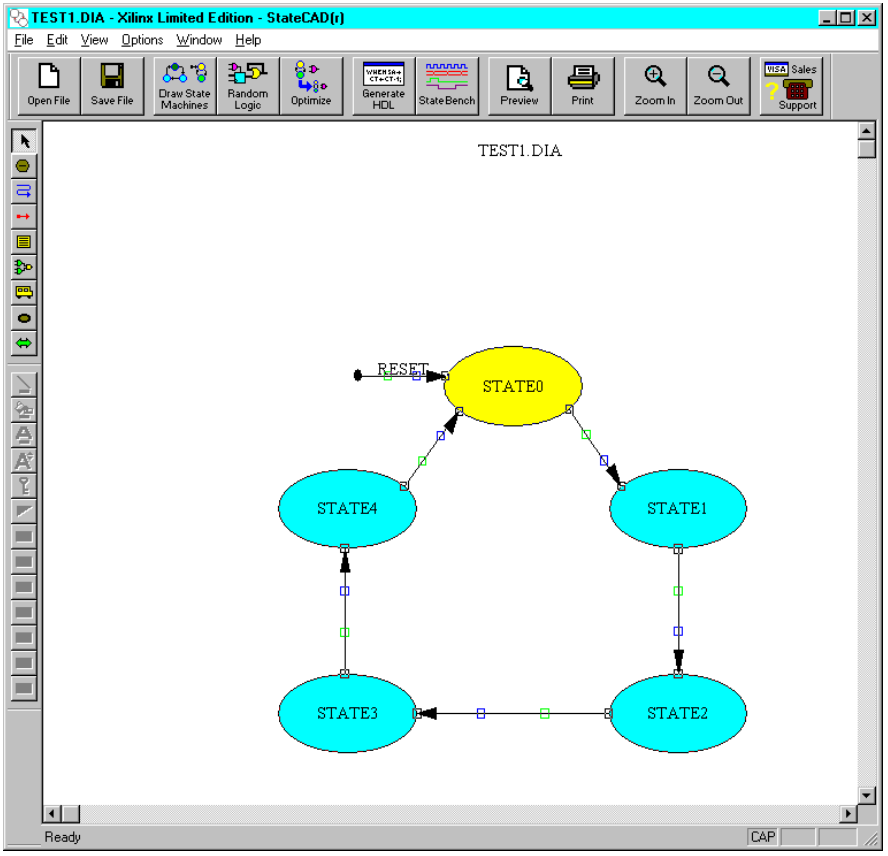
Note The name entered here is used by StateCAD in the module/entity definition in the HDL code it creates from the state diagram.

4. By default, StateCAD saves the new source in the project directory. Specify a different directory in the Location field on the New source window, if desired.
5. The "Add to Project" check box on the New source window has no effect for state diagrams. State Diagrams currently cannot be automatically added to a Foundation Series ISE project. Refer to the "Adding State Diagram Sources to Your Project" section for information on adding state diagram source files to your project.
6. Press **Next** when you are ready to continue. The New Source Information window, which summarizes the requested information for the new source, appears.
7. Press **Finish** at the New Source Information screen to invoke StateCAD.

   When StateCAD is invoked, Foundation Series ISE passes project-specific information to StateCAD, such as the project directory, synthesis tool selection, targeted device, and encoding setting.
8. The StateCAD main window appears (see the following figure) with the newly specified state diagram loaded and ready for you to begin designing the state machine.



9. Create the state diagram. The state diagram (.dia file) is written to the source directory for your project by default.You can add the state diagram to the project as a user document, if desired; refer to the “Adding State Diagram Sources to Your Project” section for details.



Following are some references and resources available to you for learning how to use the StateCAD state diagram editor:

- StateCAD online help

  If you need help using StateCAD, refer to the StateCAD online help.

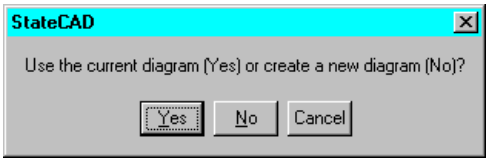
- StateCAD tutorial

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

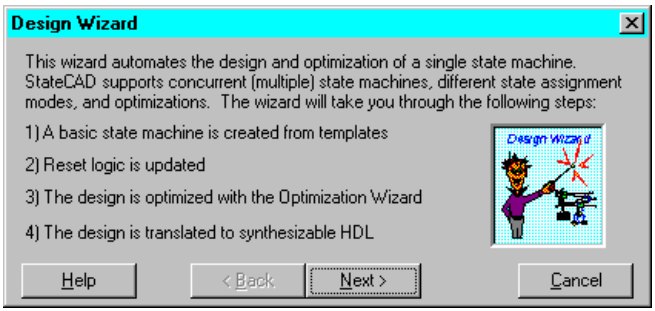
- Design Wizards

  Select **`File`** → **`Design Wizard`** from the StateCAD menu to initiate the Design Wizard.

If you use this option, a dialog box appears to have you verify whether you want to use the currently opened state diagram or a new one. Click **Yes**.

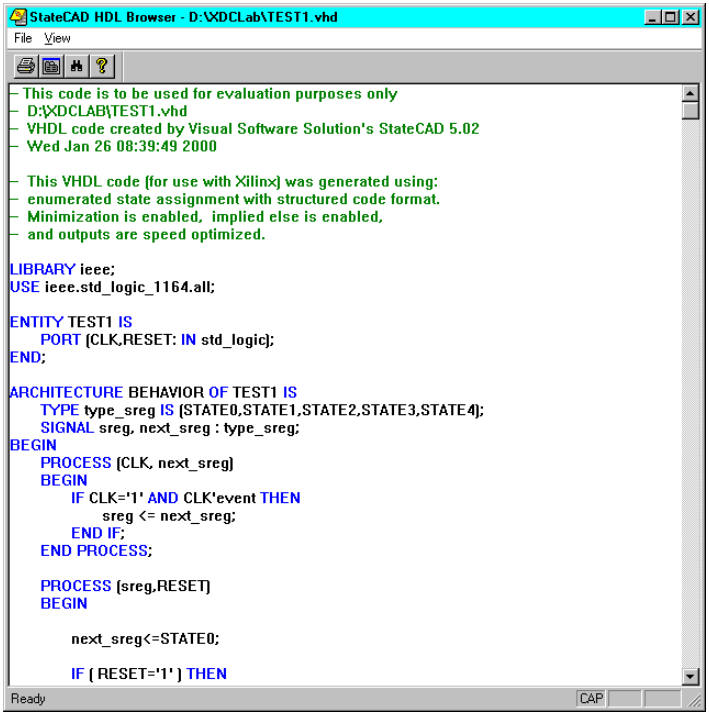


The StateCAD Design Wizard leads you through the design process. Project information (such as synthesis tool, targeted device) is reflected in the Design Wizard dialog boxes.



10. When the state diagram and its logic is valid, StateCAD automatically generates an HDL file (VHDL, Verilog, or ABEL-HDL as appropriate for your project) with the same name as the state diagram file except the file extension will be one of the following: .vhd for VHDL, .v for Verilog, or .abl for ABEL-HDL.

    For example, for a VHDL project and a state diagram file named test1.dia, StateCAD produces the VHDL file named TEST1.vhd. Within the VHDL file, the entity is defined as TEST1.



The HDL file is written to the source directory for your project. You can add the HDL file to the project as a source, if desired. Refer to the "Adding State Diagram Sources to Your Project" section for details.

Updating an Existing State Diagram

Existing StateCAD state diagrams (.dia files) can be opened in StateCAD from the Project Navigator using either of the following methods:

- Select **Project** → **New Source** from the Project Navigator. When the New screen dialog box appears, select **State Diagram** and enter the name of the desired state diagram (without the extension). Press **Next**. Then press **Finish** at the New Source Information window to open StateCAD with the specified state diagram.

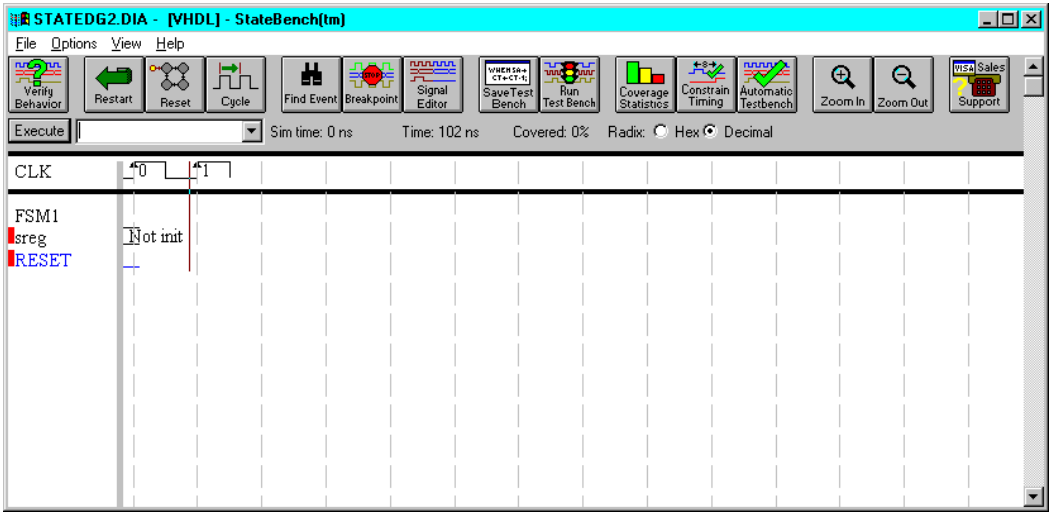
- If the state diagram is added to the project, double-click on its name in the Project Navigator's Source window. The state diagram is opened in StateCAD immediately.

  Refer to "Adding State Diagram Sources to Your Project" section for information on adding the .dia file to a project.

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 to the project, it is automatically overwritten with the updated version.

Using StateBench

Within StateCAD, you may also invoke StateBench by selecting **`Options`** → **`StateBench (Create Test Bench)`**. You can use StateBench to create test benches (VHDL) and test fixtures (Verilog) to verify the StateCAD design's behavior and validate its timing.



You can also add and use StateBench testbenches in your project.

If you need help using StateBench, refer to the StateBench online help. A StateBench tutorial is also available from the online help (**`Help`** → **`Tutorial`**).

Adding State Diagram Sources to Your Project

You must manually add files created by StateCAD to your Foundation Series ISE project. Only the StateCAD-generated synthesizable HDL file can be added as a source in your project. The user-generated state diagram can be added to the project as a user document to provide quick access to StateCAD for modification.

After the StateCAD-generated HDL file has been added to your project, it can be updated automatically when changes are made in StateCAD to the state diagram from which it was produced.

If you modify the HDL file directly, the state diagram and HDL file become out of sync. If you then modify the state diagram in StateCAD, the changes you make the to HDL file are overwritten by the new StateCAD-generated HDL file.

User-Entered State Machine Diagram

StateCAD produces a .dia file for user-created state diagrams. You can add the .dia file to your project as a user document as follows:

1. Create and save the state diagram (.dia file) in StateCAD. By default, StateCAD saves state diagram files to the source directory for the project.
2. From the Project Navigator menu select **Project** → **Add Source**.
3. Select “State Diagram” from the “Files of type” pull down menu to access the .dia file.
4. Click on the desired .dia file to select it.
5. Click **Open** to add the file to the project as a user document.
6. The file appears in the user document section of the Source window (above the device / synthesis tool line).

StateCAD-Generated HDL File

When the state diagram and its logic is valid, StateCAD generates an HDL file (VHDL, Verilog, or ABEL-HDL as appropriate for your project) with the same name as the state diagram file. The file extension will be one of the following: .vhd for VHDL, .v for Verilog or .abl for ABEL-HDL. For example, for a VHDL project and a state diagram

file named statem1.dia, StateCAD produces a VHDL file named STATEM1.vhd.

By default, the StateCAD-generated HDL file is written to the source directory for your project. You can add the HDL file to your project as a source as follows:

1. From the Project Navigator menu, select **Project** → **Add Source**.
2. Select "Sources" from the "Files of type" pull down menu to access the StateCAD-generated HDL file.
3. Click on the desired HDL file to select it.
4. Click **Open** to add the file to the project as a source.
5. The file appears in the Source window (below the device / synthesis tool line).

Note After you add a state diagram HDL file to your project, you should use StateCAD if you want to make changes to it and keep it in sync with the state diagram .dia file. StateCAD automatically updates the HDL file when you make changes to the state diagram (.dia file). If you use Project Navigator's HDL Editor to edit the HDL file, changes you make there are unknown to StateCAD.

Instantiating State Diagram Modules in HDL Designs

You instantiate the generated HDL fils as you would any HDL program.

StateCAD generates two module / entities. The first entity is a bit-wise description with the entity name SHELL_*FILENAME*. The second entity called *FILENAME* is the one to use. *FILENAME* is the name of the .dia source file. The *FILENAME* entity maps all bits into the vectors used within the design.

Instantiating State Diagram Modules in Schematic Designs

You instantiate the generated HDL fils as you would any HDL component.

StateCAD generates two module / entities. The first entity is a bit-wise description with the entity name SHELL_*FILENAME*. The second entity called *FILENAME* is the one to use. *FILENAME* is the name of the .dia source file. The *FILENAME* entity maps all bits into the vectors used within the design.

Using Foundation Series 2.1i State Diagrams

To use state diagrams created in Foundation Series 2.1i projects in Foundation Series ISE 3.1i projects, you must first convert them for StateCAD. StateCAD has an import function for this purpose.

Use the following procedure to import Foundation Series 2.1i state diagrams into StateCAD:

1. Open StateCAD.
2. Select **`File`** → **`Import Foundation Diagram (.ASF)`** from the StateCAD main menu.
3. StateCAD displays dialog boxes containing choices you need to make for the conversion.

For the latest support information on importing Foundation Series 2.1i Finite State Machines into StateCAD, go to the StateCAD website `http:/www.statecad.com` and choose the **`Support`** option.

Chapter 8

LogiBLOX

LogiBLOX is an on-screen design tool for creating high-level modules such as counters, shift registers, and multiplexers for XC4000, Spartan, and SpartanXL FPGA designs and any 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, high-level LogiBLOX modules that will fit into your schematic-based design or HDL-based design can be created and processed.

This chapter contains the following sections.

- "Accessing LogiBLOX"
- "LogiBLOX Setup"
- "Creating LogiBLOX Modules"
- "Using LogiBLOX Modules in ISE Projects"
- "Simulating LogiBLOX Components"
- "Constraining LogiBLOX RAM/ROM with FPGA Express"
- "Documentation"

Note LogiBLOX is not available for use with Virtex, VirtexE, Virtex2, and Spartan2 devices. The CORE Generator supports those devices.

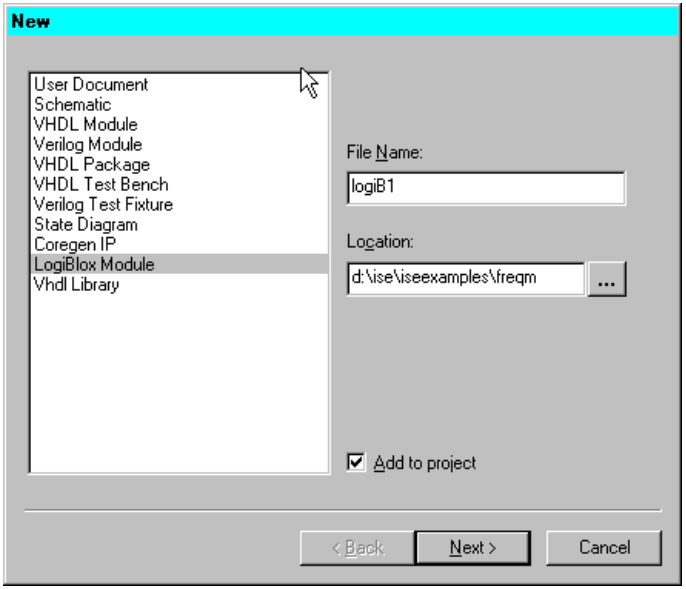
Accessing LogiBLOX

In Foundation Series ISE, you can access LogiBLOX in two ways: as a standalone tool or as an integrated design entry tool in the Project Navigator.

You can access LogiBLOX as a standalone tool by selecting **Start** → **Foundation Series ISE 3.1i** → **Accessories** → **LogiBLOX** from your PC's desktop.

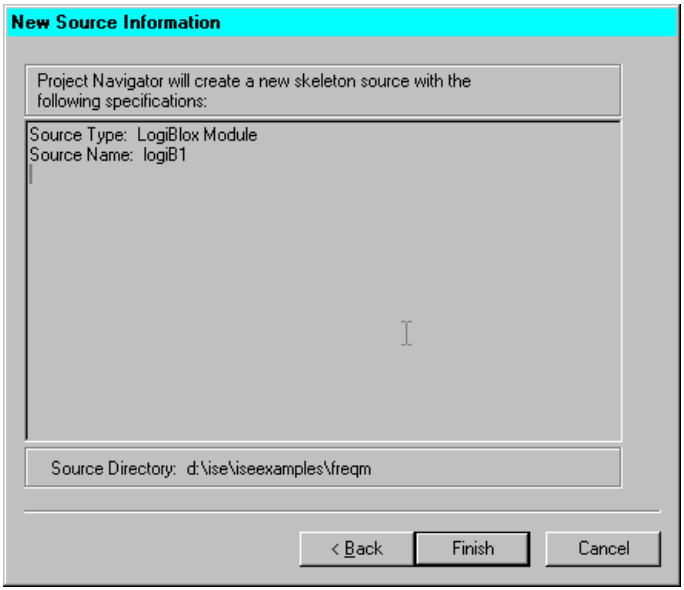
Within a Foundation Series ISE project, LogiBLOX is an integrated design entry tool for projects targeting XC4000, Spartan, SpartanXL, or CPLD devices. You can access it to create a CORE for use in a design as follows.

1. Select **Project** → **New Source** from the Project Navigator menu.
2. Select **LogiBlox Module** from the New source window.



3. Enter a File Name for the new module. Then click **Next** to continue.

4. At the New Source Information window, click **Finish**.



5. The LogiBLOX tool is invoked and opens. Its initial screen is shown in the following figure.

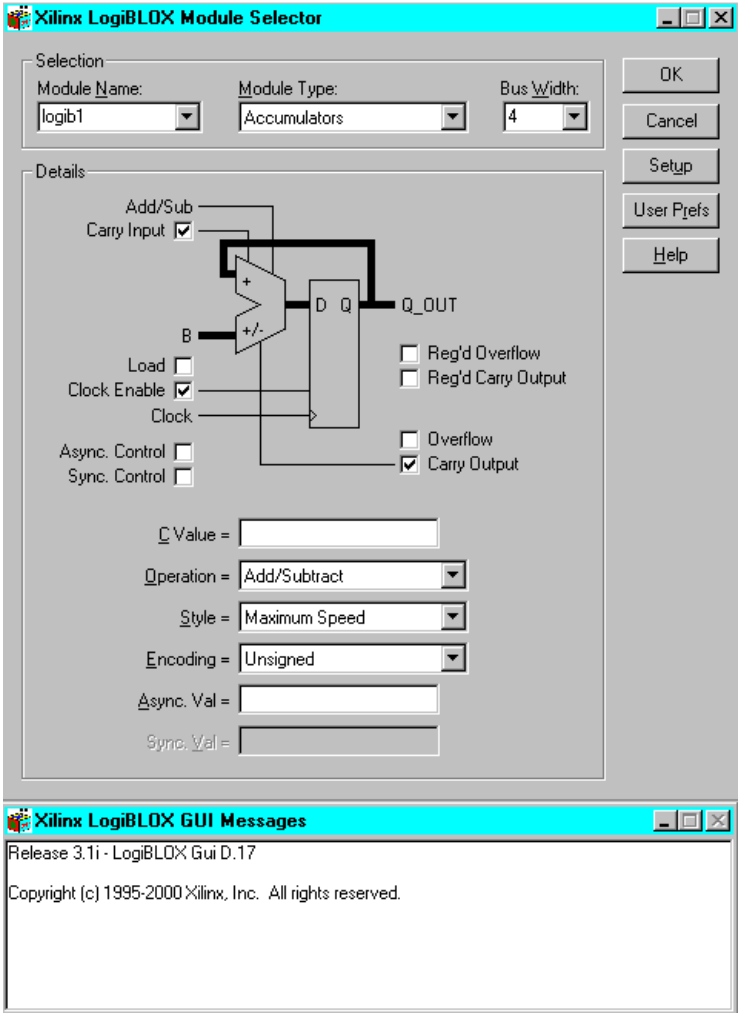
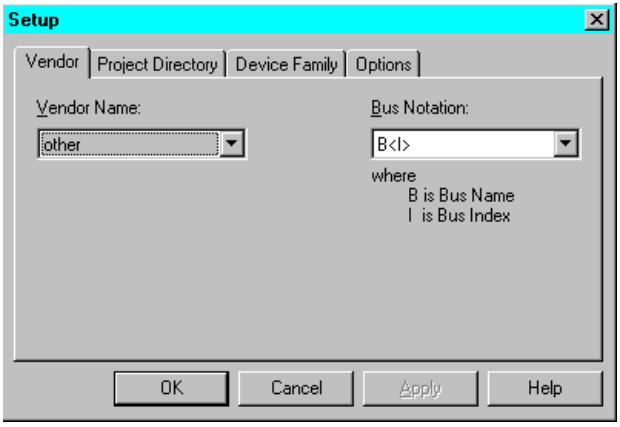


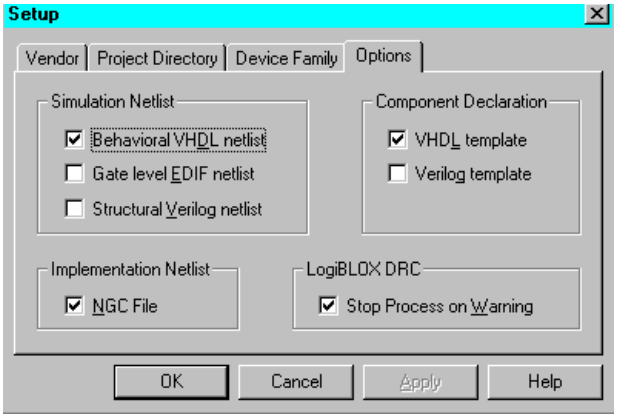
Figure 8-1 LogiBlox Module Selector - Accumulators

LogiBLOX Setup

When you access LogiBLOX from a Foundation Series ISE 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 access the LogiBLOX Setup dialog box (shown in the following figure), 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. You can change this by using the Options tab to select the type of simulation netlist and component declaration template (VHDL or Verilog) you need. For VHDL, select **VHDL template** and **Behavioral VHDL netlist** (shown below). For Verilog, select **Verilog template** and **Structural Verilog netlist**.



Creating LogiBLOX Modules

This section contains a very brief overview of the LogiBLOX tool. Refer to the *LogiBLOX Guide* and LogiBLOX's online Help for detailed information on LogiBLOX.

After you have opened LogiBLOX from your Foundation Series ISE project, create a new module as follows:

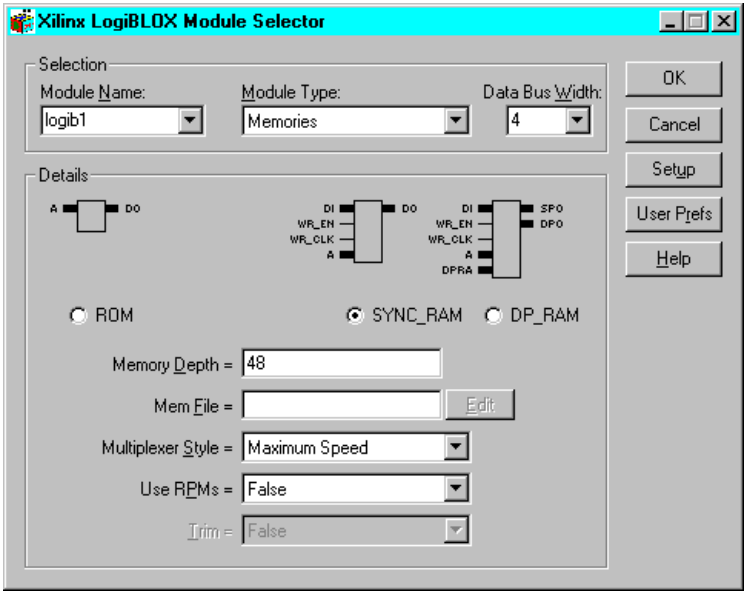
1. The name you entered in the Project Navigator's New source appears in the Module Name field. You need not enter a name.

   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 Figure 8-1). Click on the down arrow in the Module type field to display the list of modules 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	Tristate Buffers

3. Select a module type and define its attributes. The following figure shows the LogiBLOX dialog boxes 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**.

When you click **OK**, the LogiBLOX module and its related files are created. The LogiBLOX .mod file is automatically added to the Foundation Series ISE project as a user document.

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

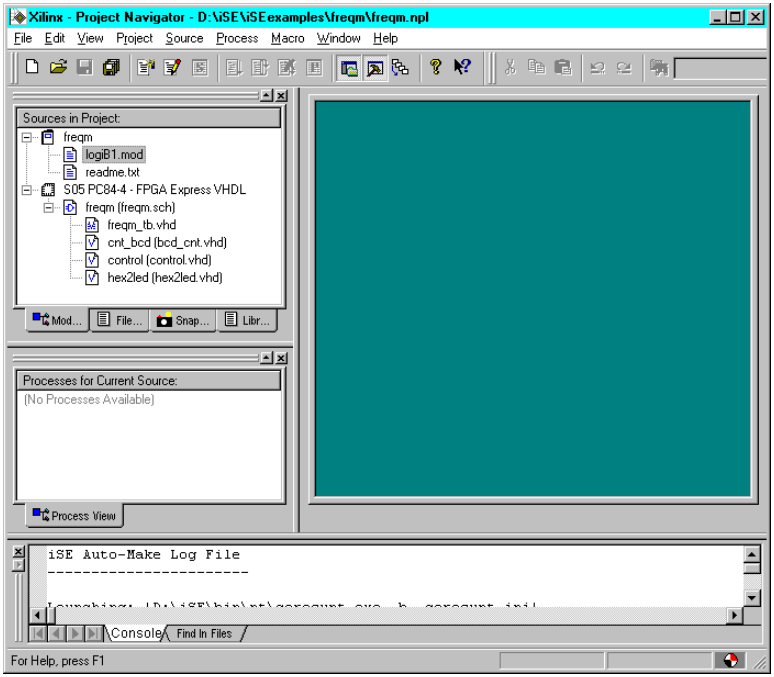
Note Whether the VHDL files (.vhi, .vhd) and / or Verilog files (.vei, .v) are created depends on the setting in the LogiBLOX Setup Options tab.

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 Projects

If you initiated LogiBLOX from the Project Navigator New source window, the LogiBLOX module and its related files are placed in the current Foundation Series ISE 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 to the project as a user document and appears in the Source window (logib1.mod in the following figure).



Editing LogiBLOX Modules

If you want to edit a LogiBLOX module in a Foundation Series ISE project, double-click on its name in the Source window. This opens the LogiBLOX Module Selection window and loads it with the selected module.

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.

The following procedure describes how to use LogiBLOX in schematic sources.

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

2. Click on the newly added *logiblox_module*.v or .vhd file in the Source window.
3. Click on **`Create Schematic Symbol`** in the Process window.
4. A symbol (*logiblox_module*.sym) is created for the *logiblox_module*.v or .vhd module.
5. The *logiblox_module* symbol is added to the local Symbol Library of the Schematic Editor for use in schematic sources.
6. 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 Foundation 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.

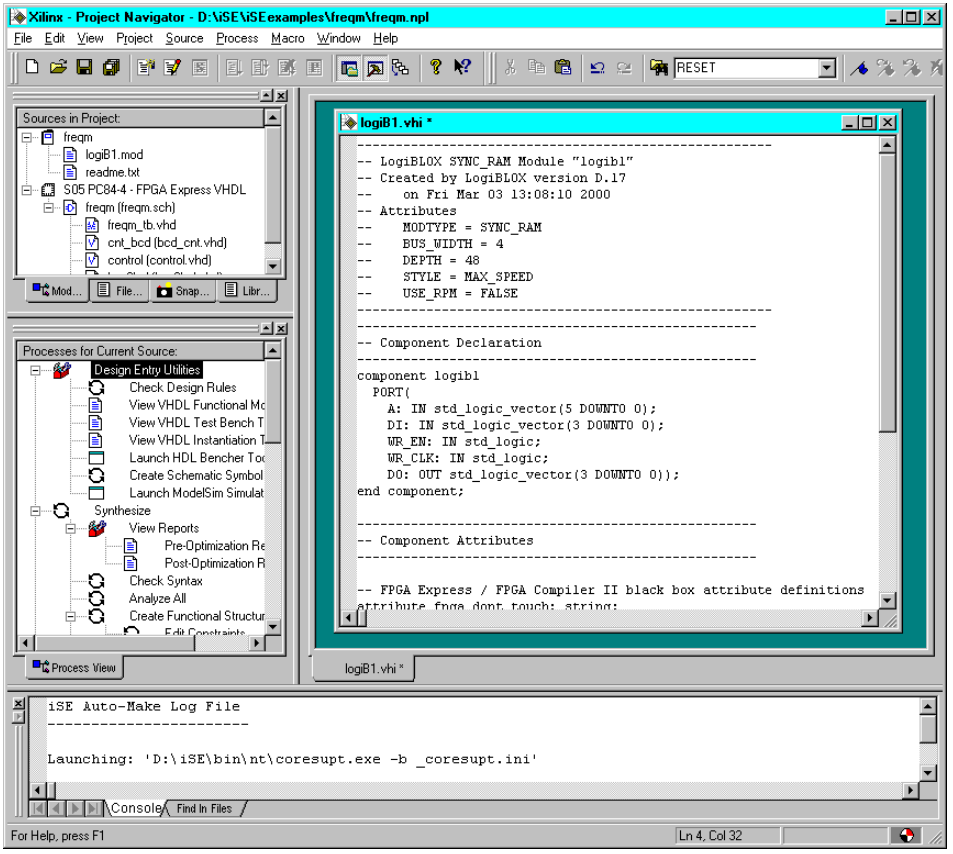
VHDL Instantiation

When you instruct LogiBLOX to create a VHDL component in a Foundation Series ISE 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.

Use the following procedure to instantiate a LogiBLOX module into a VHDL source in a Foundation Series ISE project. The example used in

this section is for a LogiBLox RAM48X4S memory component named “logiB1.”

1. Select **File** → **Open** from the Project Navigator menu.
2. When the Open dialog box appears, select the VHDL instantiation template (.vhi file) for the desired LogiBLOX module from the project directory.
3. The VHDL instantiation template opens in the HDL Editor workspace. An example for a LogiBLOX memory module named “logib1” is shown in the following figure.



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 => );
```

4. Use **File** → **Open** on the Project Navigator menu to open the VHDL file in which the LogiBLOX component is to be instantiated.
5. 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.
6. Cut and past 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.

An example of VHDL source code with the LogiBLOX instantiation for the component named *logib1* is shown below. 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;

7. 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.

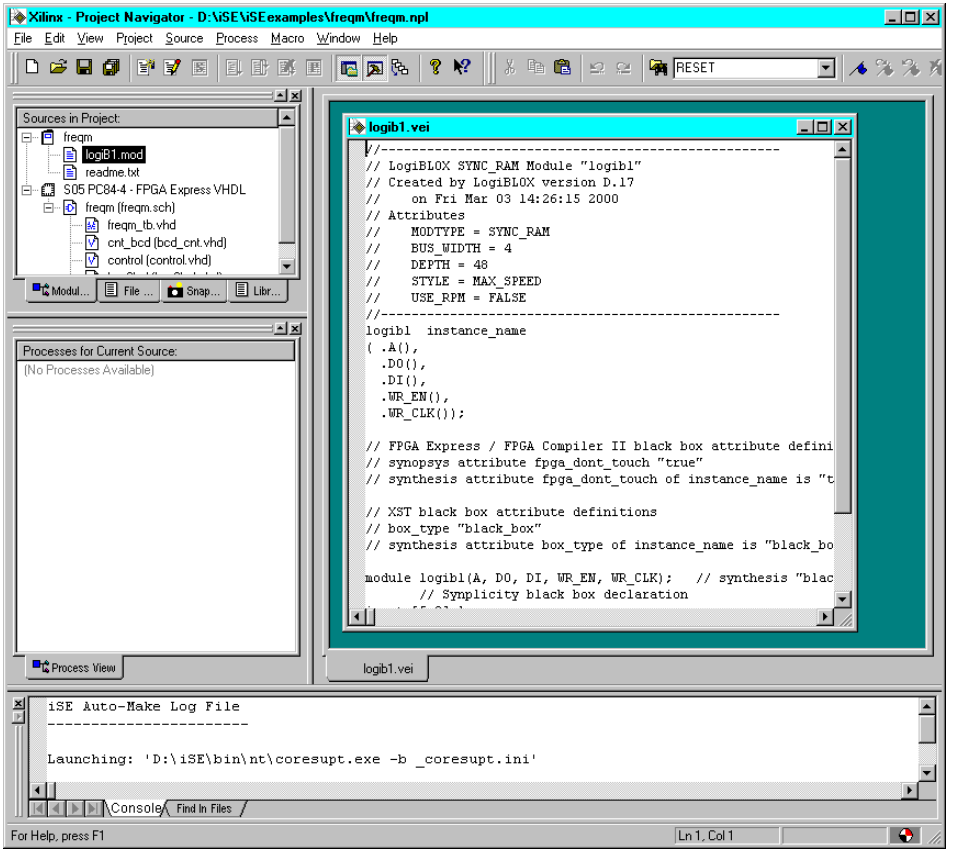
Verilog Instantiation

When you instruct LogiBLOX to create a Verilog component in a Foundation Series ISE 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 a Foundation Series ISE project. The example used in this section is for a LogiBLOX RAM48X4S memory component named “logib1.”

1. Select **File** → **Open** from the Project Navigator menu.

2. When the Open dialog box appears, select the Verilog instantiation template (.vei file) for the desired LogiBLOX module from the project directory.
3. The VHDL instantiation template opens in the HDL Editor workspace. An example for a LogiBLOX memory module named “logib1” is shown in the following figure.



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
```

4. Use **File** → **Open** on the Project Navigator menu to open the Verilog file in which the LogiBLOX component is to be instantiated.
5. Cut and paste the module declaration from the LogiBLOX component's .vei file into the Verilog design code, placing it after the "endmodule" line within the architecture section or the Verilog design code.
6. 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.

   An example of Verilog source code with the LogiBLOX instantiation for the component named *logib1* is shown below. 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
```

7. 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, you need to add the *component_name*.vhd file created by LogiBLOX to the project. For Verilog components, this step is not necessary.

Constraining LogiBLOX RAM/ROM with FPGA Express

In the XSI (Xilinx Synopsys Interface) HDL methodology, whenever large blocks of RAM/ROM are needed, LogiBLOX RAM/ROM modules are instantiated in the HDL code. With LogiBLOX RAM/ROM modules instantiated in the HDL code, timing and/or placement constraints on these RAM/ROM modules, and the RAM/ROM primitives that comprise these modules, can be specified in a UCF file. To create timing and/or placement constraints for RAM/ROM LogiBLOX modules, knowledge of how many primitives will be used and how the primitives, and/or how the RAM/ROM LogiBLOX modules are named is needed.

Estimating the Number of Primitives Used

When a RAM/ROM is specified with LogiBLOX, the RAM/ROM depth and width are specified. If the RAM/ROM depth is divisible by 32, then 32x1 primitives are used. If the RAM/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 RAM/ROM can be calculated.

For example, if a RAM48x4 was required for a design, RAM16x1 primitives would be used. Based on the width, there would be four banks of RAM16x1s. Based on the depth, each bank would have 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 RAM/ROM module has an instance name of MEMx_y, where y represents the primitive position in the bank of memory and where x represents the bit position of the RAM/ROM output.

For the next two items, refer to the Verilog/VHDL examples included at the end of this section. The Verilog/VHDL example instantiates a RAM32x2S, which is in the bottom of the hierarchy. The RAM32x2S was 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/Component in an HDL Source

LogiBLOX RAM/ROM modules in the HDL Flow are constrained via a UCF file. LogiBLOX RAM/ROM modules instantiated in the HDL code can be referenced by the full-hierarchical instance name. If a LogiBLOX RAM/ROM module is at the top-level of the HDL code, then the instance name of the LogiBLOX RAM/ROM module is just the instantiated instance name.

In the case of a LogiBLOX RAM/ROM, which is instantiated within the hierarchy of the design, the instance name of the LogiBLOX RAM/ROM module is the concatenation of all instances which contain the LogiBLOX RAM/ROM. The concatenated instance names

are separated by a "_". In the example, the RAM32X1S is named `memory`. The module `memory` is instantiated in Verilog module `inside` with an instance name U0. The module `inside` is instantiated in the top-level module test. Therefore, the RAM32X1S can be referenced in a .ucf file as U0/U0. For example, to attach a TNM to this block of RAM, the following line could be used in the UCF file.

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

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

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 RAM/ROM 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 RAM/ROM module.

Returning to the RAM32x2S example above, suppose that the each of the RAM primitives had to be constrained to a particular CLB location. Based on the rules for determining the MEMx_y instance names and using the example from above, each of the RAM primitives could be referenced 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;
```

Documentation

The following documentation is available for the LogiBLOX program:

- The *LogiBLOX Guide* is available with the Xilinx online book collection on the CD-ROM supplied with your software or from the Xilinx web site at `http://support.xilinx.com`.
- You can access LogiBLOX online help from LogiBLOX or from the Foundation online help system.
- The *Xilinx Software Conversion Guide from XACTstep v5.X.X to XACTstep vM1.X.X* compares XBLOX and LogiBLOX. It describes how to convert an XBLOX design to LogiBLOX. This document is available on the Xilinx web site at `http://support.xilinx.com`.

Chapter 9

CORE Generator

The Xilinx CORE Generator System is a design tool that delivers parameterizable COREs optimized for Xilinx FPGAs. It provides the user 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 including filters, transforms, memories.

New COREs can be downloaded from the Xilinx web site and added to the CORE Generator System. The URL for downloading CORES is `http://www.xilinx.com/ipcenter`. You can check this web site to verify you have the latest version of each CORE and CORE data sheet.

This chapter contains the following sections:

- "Accessing the CORE Generator System"
- "Creating a CORE Component"
- "Using COREs in Foundation Series ISE Projects"
- "Simulation and Synthesis of CORE Modules"
- "Simulating COREs in a Schematic"

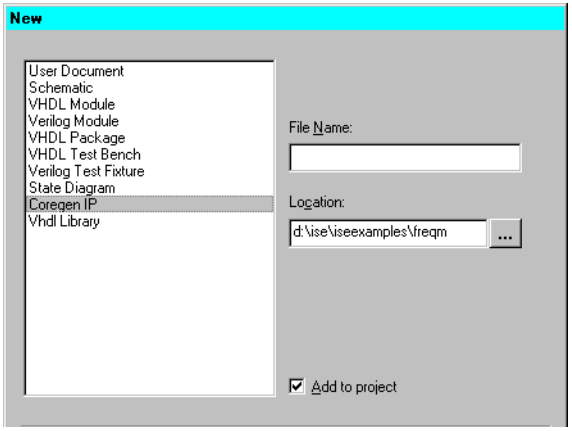
Accessing the CORE Generator System

In Foundation Series ISE, you can access the CORE Generator System in two ways: as a standalone tool or as an integrated design entry tool in the Project Navigator.

You can access the CORE Generator as a standalone tool by selecting **`Start`** → **`Foundation Series ISE 3.1i`** → **`Accessories`** → **`CORE Generator`** from your PC's desktop.

Within a Foundation Series ISE project, the CORE Generator is an integrated design entry tool. You can access it to create a CORE for use in a design as follows.

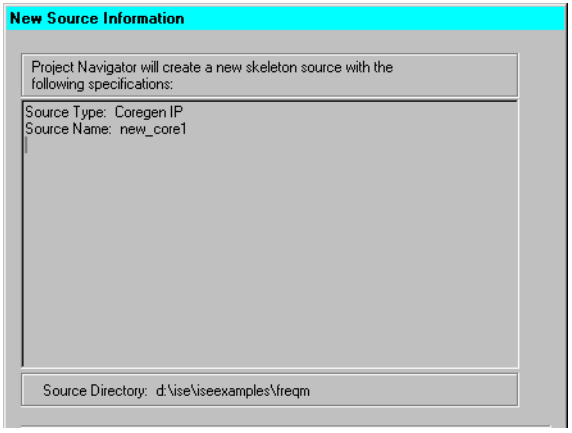
1. Select **Project** → **New Source** from the Project Navigator menu.
2. Select **Coregen IP** from the New source window.



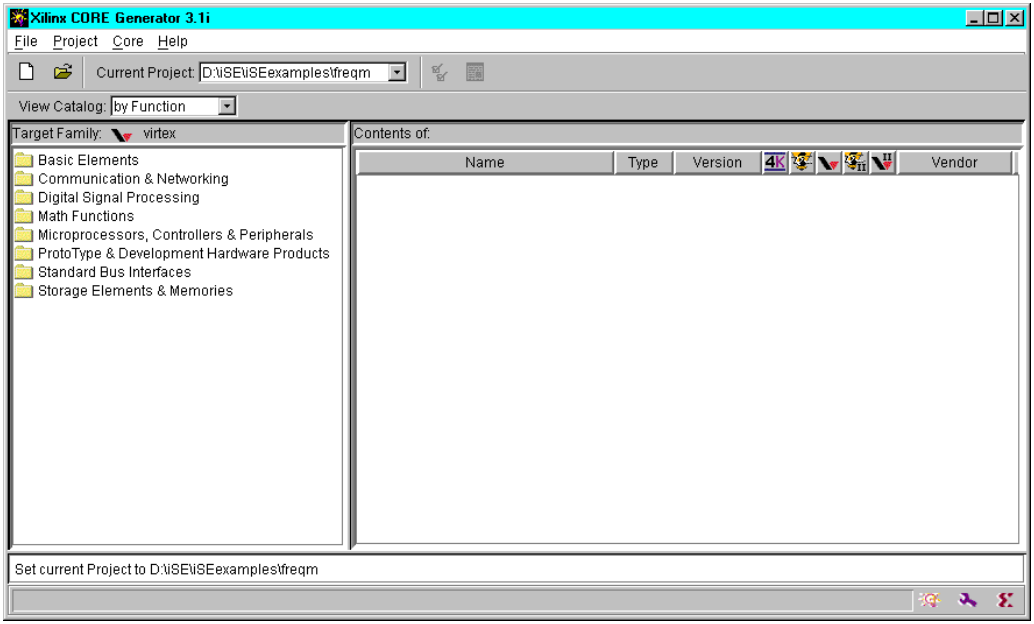
3. Enter a File Name for the new CORE. Then click **Next** to continue.

   Note Currently, the file name entered here is not passed to the CORE Generator.

4. At the New Source Information window, click **Finish**.



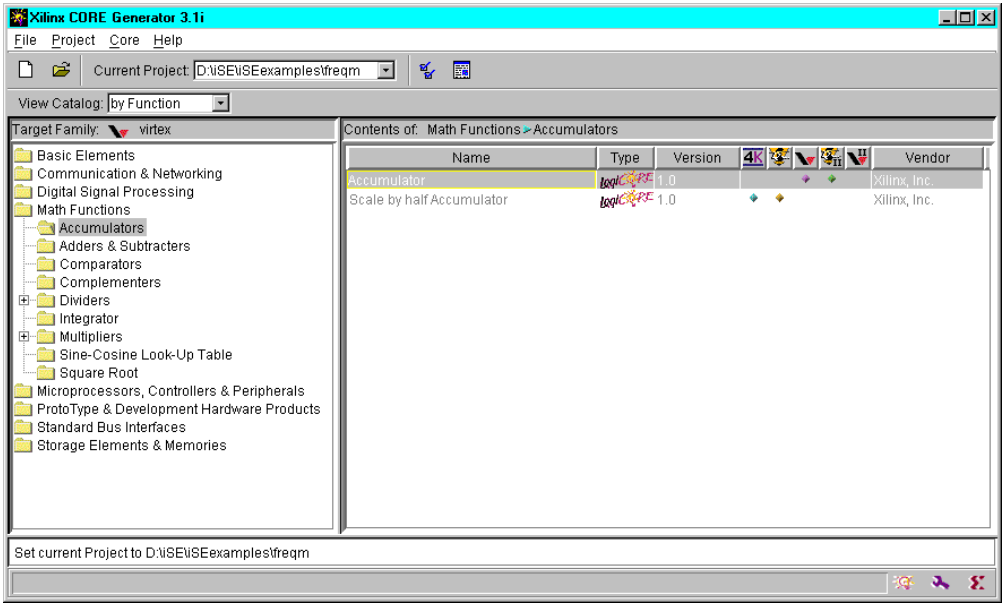
5. The CORE Generator tool is invoked and opens. Its initial screen is shown in the following figure.



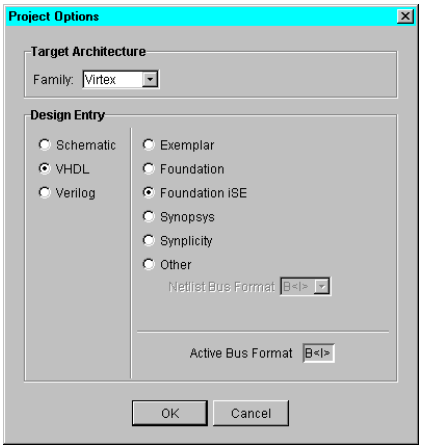
Creating a CORE Component

This section contains a very brief overview of the CORE Generator tool. You can access detailed documentation and CORE Generator support websites from the CORE Generator **Help** menu.

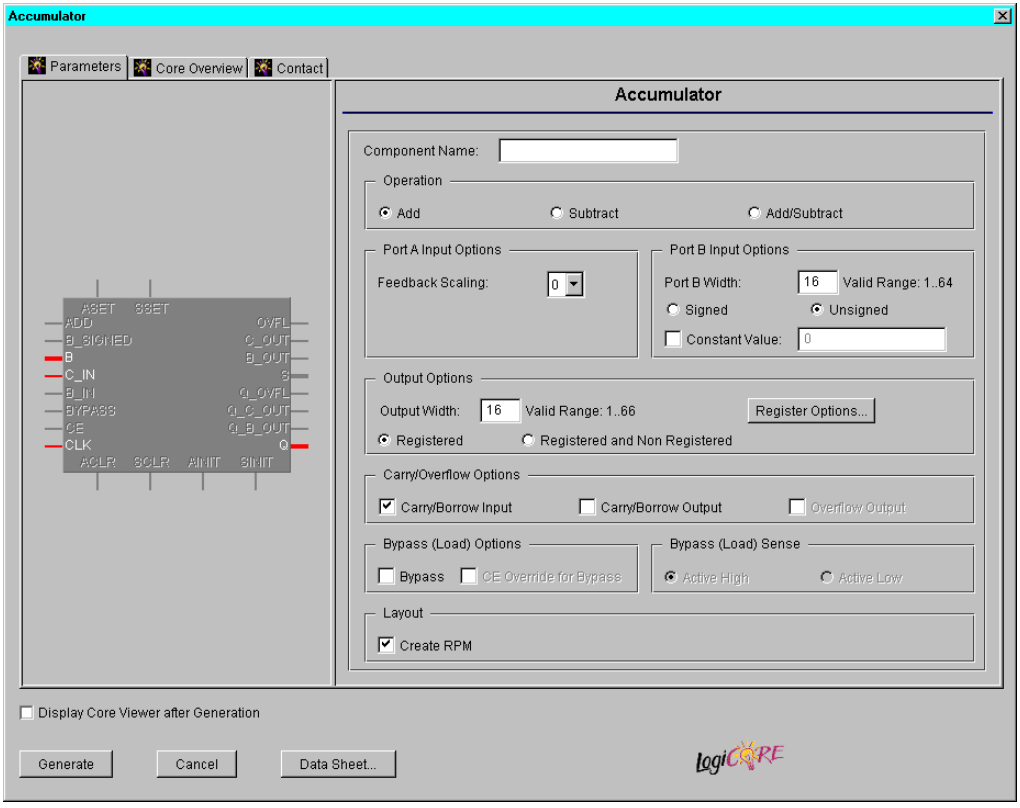
The initial Xilinx CORE Generator window allows selection of the available COREs. The COREs are categorized on the left side of the window. The specific COREs are selected in the “Contents of” section of the window as shown in the following figure.



You can select **`Project`** → **`Project Options`** from the CORE Generator menu to access the project setup options. However, if you accessed the CORE Generator from a Foundation Series ISE project, the Project Options are automatically set to the appropriate values for the project. You do not need to set them manually—except for the new component name entered in the New source dialog box in the Project Navigator.



You select a CORE by double-clicking on its name in the “Contents of” section of the CORE Generator window. This opens a new window 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 following figure shows the CORE Generator window that appears when you select the accumulator CORE for a Virtex project from the initial screen.



Note The Component Name field is blank on the CORE Generator customization screens.The name entered in the Foundation Series ISE Project Navigator New source window is not passed to the CORE Generator tool.

You must manually enter a name in the Component Name field. CORE component names have the following requirements:

- The name must begin with an alpha character.
- No extensions or uppercase letters are allowed.
- After the first character, the name may include numbers and/or the underscore character.

Click the Data Sheet button to view detailed information on the CORE. You must have the Adobe Acrobat Reader installed on your PC to view the data sheet.

After you customize the CORE for your needs, you need to generate the new CORE (click the **Generate** button). After the CORE has been successfully generated, the new CORE and its related files are placed in the project directory. When finished, close the CORE Generator windows.

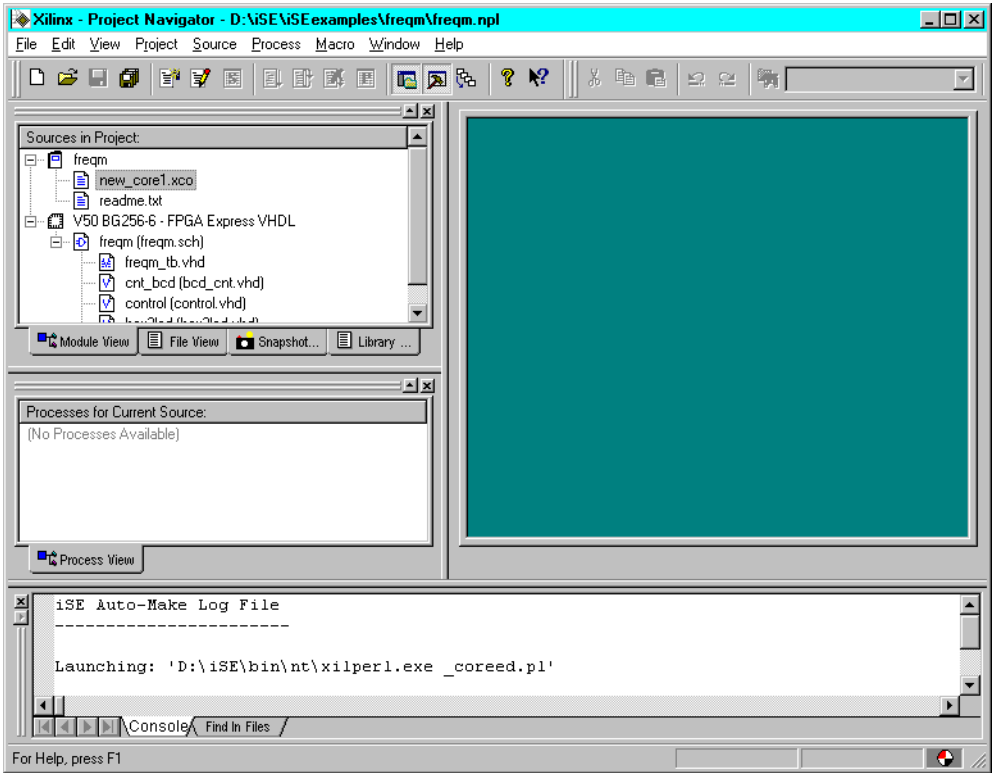
The customized CORE component is a collection of several files including those listed below. The files are located in your Foundation Series ISE project directory.

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

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 CORE s in Foundation Series ISE Projects

If you initiated the CORE Generator tool from the Project Navigator New source window, the new CORE and its related files are placed in the current Foundation Series ISE project directory. The related files include schematic, VHDL, and Verilog templates that can be used in your project. The CORE is added to the project as a user document and appears in the Source window (new_core1.xco in the following figure).



Editing CORE s

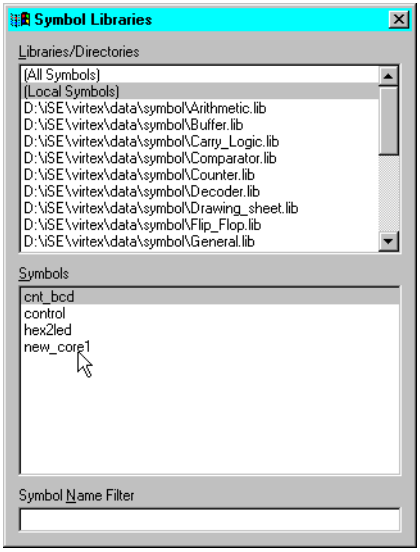
If you want to edit a CORE in a Foundation Series ISE project, double-click on its name in the Source window. This opens the CORE Generator's customization window for that CORE. When you regenerate it, the files related to that CORE are updated. The following five file types are related to CORE s: .sym, .veo, .vho, .xco, .edn.

Using COREs in Schematic Sources

The CORE Generator automatically creates a schematic symbol for any CORE component. When you create a CORE in a Foundation Series ISE project, its schematic symbol is automatically added to the ECS Schematic Editor's local symbol library for use in schematic designs for that project.

Use the following procedure from your Foundation Series ISE project to select a CORE for use in a schematic in that project.

1. Open the ECS Schematic Editor either by double-clicking on an existing schematic source in the Source window. Or, select **`Project`** → **`New source`** and then **`Schematic`** to create a new schematic source.
2. When the Schematic Editor is open, select **`Add`** → **`Symbol`** from the Schematic Editor menu.
3. Then select the desired CORE from the Local Symbol library for placement on the open schematic. An example (new_core1) is shown in the following figure.

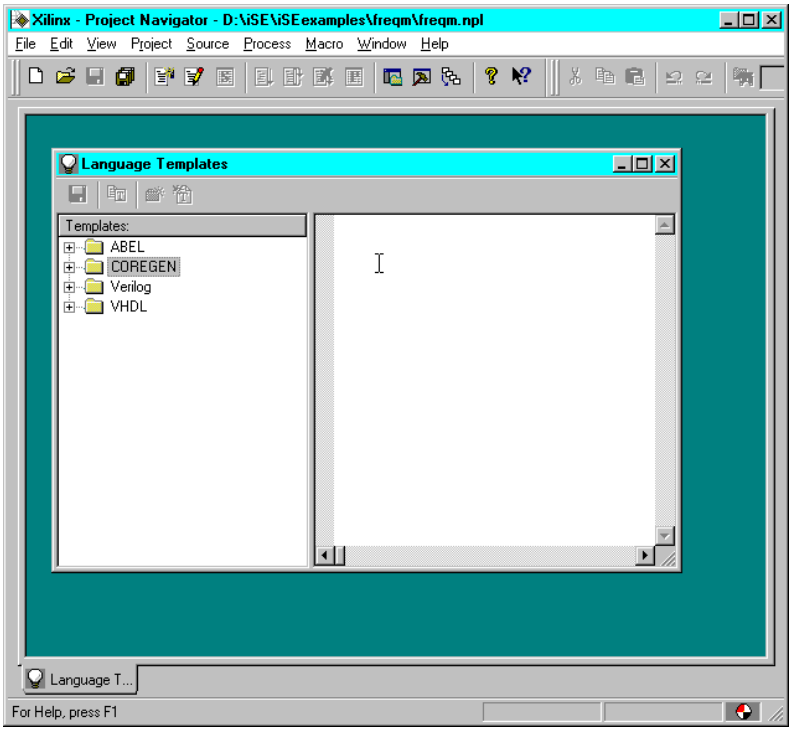


Instantiating COREs in an HDL Source

The CORE Generator automatically creates a VHDL template and Verilog template for each CORE component. When you create a CORE in a Foundation Series ISE 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.

Use the following procedure to access the CORE component HDL instantiation templates for a project.

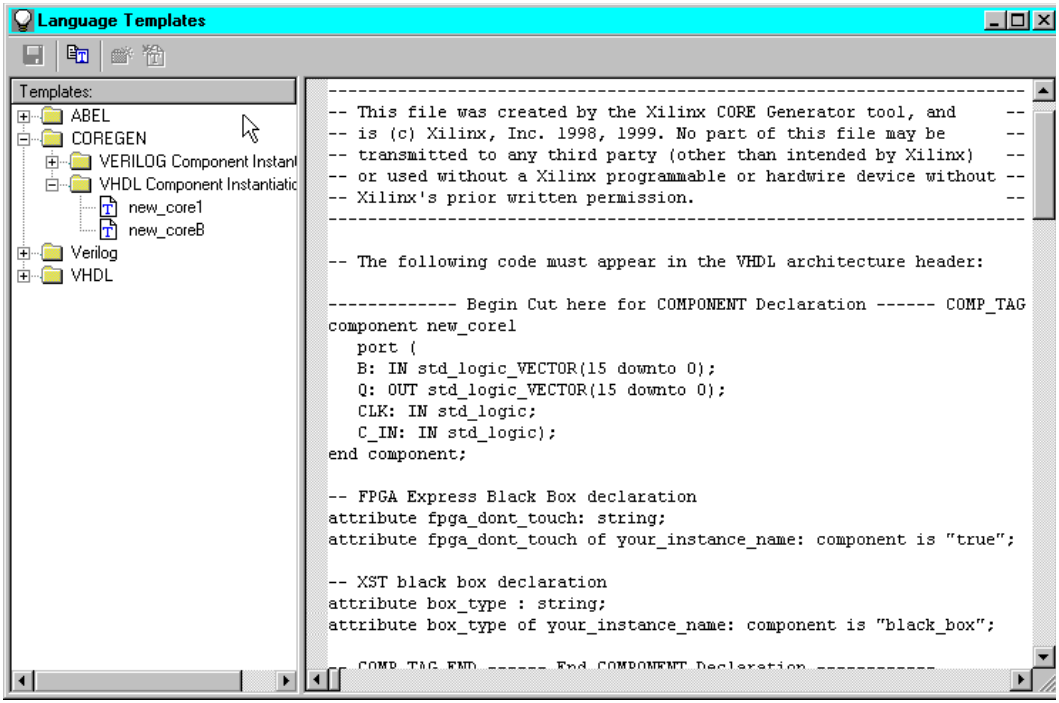
1. Select **Edit** → **Language Templates** from the Project Navigator menu.
2. The Language Template window opens in the HDL Editor workspace as shown in the following figure.



3. Click on the "+" icon beside **COREGEN** in the Language Templates window to access the folders under COREGEN.

4. For VHDL projects, click on the "+" icon beside **VHDL Component Instantiation** folder to display the list of available VHDL instantiation templates. For Verilog projects, click on the "+" icon beside **Verilog Component Instantiation**.

5. Double click on the desired CORE to display its instantiation template in the right pane of the Language Templates window. An example is shown in the following figure.



6. To use the template, cut and paste the template text from the Language Templates window to a new or open HDL file.

VHDL Instantiation Template Example

Each COREGEN VHDL instantiation template in the Language Templates tool contains complete instructions on how to instantiate it in VHDL code. The components are instantiated as black boxes.

Note When the design is synthesized with FPGA Express, a warning is generated that the CORE module is unexpanded. (Modules instan-

tiated as black boxes are not elaborated and optimized. The warning message is just reflecting the black box instantiation.

The following code is an example of a VHDL instantiation template for a CORE accumulator component.

```
------------------------------------------------------------
-- This file was created by the Xilinx CORE Generator tool, and    --
-- is (c) Xilinx, Inc. 1998, 1999. No part of this file may be     --
-- transmitted to any third party (other than intended by Xilinx) --
-- or used without a Xilinx programmable or hardwire device without-
-- Xilinx's prior written permission.                              --
------------------------------------------------------------------

-- The following code must appear in the VHDL architecture header:

--------- Begin Cut here for COMPONENT Declaration ------ COMP_TAG
component new_core1
  port (
  B: IN std_logic_VECTOR(15 downto 0);
  Q: OUT std_logic_VECTOR(15 downto 0);
  CLK: IN std_logic;
  C_IN: IN std_logic);
end component;

-- FPGA Express Black Box declaration
attribute fpga_dont_touch: string;
attribute fpga_dont_touch of your_instance_name: component is
"true";

-- XST black box declaration
attribute box_type : string;
attribute box_type of your_instance_name: component is "black_box";

-- COMP_TAG_END ------ End COMPONENT Declaration ------------

-- The following code must appear in the VHDL architecture
-- body. Substitute your own instance name and net names.

----------- Begin Cut here for INSTANTIATION Template ----- INST_TAG
your_instance_name : new_core1
    port map (
```

```
      B => B,
      Q => Q,
      CLK => CLK,
      C_IN => C_IN);
-- INST_TAG_END ------ End INSTANTIATION Template ------------

-- The following code must appear above the VHDL configuration
-- declaration. An example is given at the end of this file.

------------ Begin Cut here for LIBRARY Declaration -------- LIB_TAG

-- synopsys translate_off

Library XilinxCoreLib;

-- synopsys translate_on

-- LIB_TAG_END ------- End LIBRARY Declaration ------------

-- The following code must appear within the VHDL top-level
-- configuration declaration. Ensure that the translate_off/on
-- compiler directives are correct for your synthesis tool(s).

----------- Begin Cut here for CONFIGURATION snippet ------ CONF_TAG

-- synopsys translate_off

  for all : new_core1 use entity XilinxCoreLib.C_ACCUM_V1_0
(behavioral)generic map(
      c_sinit_val => "0",
      c_sync_enable => 0,
      c_has_ainit => 0,
      c_sync_priority => 1,
      c_b_type => 1,
      c_has_c_out => 0,
      c_scale => 0,
      c_has_sinit => 0,
      c_has_c_in => 1,
      c_has_b_out => 0,
      c_ainit_val => "0000",
```

```
      c_b_width => 16,
      c_add_mode => 0,
      c_has_sset => 0,
      c_bypass_low => 0,
      c_has_q_ovfl => 0,
      c_bypass_enable => 0,
      c_has_ovfl => 0,
      c_has_s => 0,
      c_has_aset => 0,
      c_has_add => 0,
      c_has_sclr => 0,
      c_has_q_c_out => 0,
      c_pipe_stages => 0,
      c_b_value => "0",
      c_out_width => 16,
      c_has_q_b_out => 0,
      c_has_b_in => 0,
      c_b_constant => 0,
      c_has_ce => 0,
      c_low_bit => 0,
      c_has_b_signed => 0,
      c_saturate => 0,
      c_has_bypass => 0,
      c_has_aclr => 0,
      c_high_bit => 15,
      c_enable_rlocs => 1);
  end for;

-- synopsys translate_on

-- CONF_TAG_END ------ End CONFIGURATION snippet ------------

-------------------------------------------------------------
-- Example of configuration declaration...
-------------------------------------------------------------
--
-- <Insert LIBRARY Declaration here>
--
-- configuration <cfg_my_design> of <my_design> is
--     for <my_arch_name>
--         <Insert CONFIGURATION Declaration here>
```

```
--     end for;
-- end <cfg_my_design>;
--
-- If this is not the top-level design then in the next level up,
-- the following text should appear at the end of that file:
--
-- configuration <cfg> of <next_level> is
--     for <arch_name>
--         for all : <my_design> use configuration <cfg_my_design>;
--           end for;
--     end for;
-- end <cfg>;
--
```

Verilog Instantiation Template Example

Each COREGEN Verilog instantiation template in the Language Templates tool contains complete instructions on how to instantiate it in Verilog code. The components are instantiated as black boxes.

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

The following code is an example of a Verilog instantiation template for a CORE accumulator component.

```
/*************************************************************
* This file was created by the Xilinx CORE Generator tool, and    *
* is (c) Xilinx, Inc. 1998, 1999. No part of this file may be     *
* transmitted to any third party (other than intended by Xilinx)  *
* or used without a Xilinx programmable or hardwire device without *
* Xilinx's prior written permission.                              *
*************************************************************/

// The following line must appear at the top of the file in which
// the core instantiation will be made. Ensure that the
// translate_off/_on compiler directives are correct for your
// synthesis tool(s)

//---------- Begin Cut here for LIBRARY inclusion --------// LIB_TAG
```

```
// synopsys translate_off

‘include "D:\iSE/verilog/XilinxCoreLib/src/C_ACCUM_V1_0.v"

// synopsys translate_on
// LIB_TAG_END ------- End LIBRARY inclusion --------------

// The following code must appear after the module in which it
// is to be instantiated. Ensure that the translate_off/_on compiler
// directives are correct for your synthesis tool(s).

//----------- Begin Cut here for MODULE Declaration -------//
MOD_TAG
module new_core2 (
  B,
  Q,
  CLK,
  C_IN);

input [15 : 0] B;
output [15 : 0] Q;
input CLK;
input C_IN;

// synopsys translate_off

  C_ACCUM_V1_0 #(
    0,
    "0000",
    0,
    0,
    0,
    1,
    "0",
    16,
    1,
    0,
    0,
    0,
    0,
    0,
```

```
    0,
    0,
    0,
    0,
    1,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    15,
    0,
    16,
    0,
    0,
    0,
    "0",
    0,
    1)
  inst (
    .B(B),
    .Q(Q),
    .CLK(CLK),
    .C_IN(C_IN));
// synopsys translate_on
endmodule
// MOD_TAG_END ------- End MODULE Declaration -------------

// The following must be inserted into your Verilog file for this
// core to be instantiated. Change the instance name and port
// connections (in parentheses) to your own signal names.
//--------- Begin Cut here for INSTANTIATION Template ---// INST_TAG
new_core2 YourInstanceName (
  .B(B),
  .Q(Q),
  .CLK(CLK),
  .C_IN(C_IN));
```

```
// FPGA Express black box declaration
// synopsys attribute fpga_dont_touch "true"
// synthesis attribute fpga_dont_touch of YourInstanceName is"true"
// XST black box declaration
// box_type "black_box"
// synthesis attribute box_type of YourInstanceName is "black_box"

// INST_TAG_END ------ End INSTANTIATION Template ---------
```

Simulation and Synthesis of CORE Modules

Refer to the *CORE Generator Guide* accessed from the CORE Generator **Help** → **Online Documentation** for detailed information on simulation and synthesis of CORE modules. Refer to the "Simulating COREs in a Schematic" section for ISE project specific information on simulating schematics that include CORE modules.

Simulating COREs in a Schematic

In order to perform functional simulation on schematics containing CORE Generator macros in a Foundation Series ISE project, you must first make some modifications to the testbench for VHDL simulation or to the symbol itself for Verilog simulation. The procedures to do this are described in the following sections.

VHDL Simulation

CORE Generator VHDL simulation models are linked to the design using configuration statements. When a CORE is created, an instantiation template, *core_name*.vho, is generated in the project directory that includes the majority of the configuration information.

In a schematic design containing a CORE Generator created macro, you must add a hierarchical configuration statement to the bottom of the testbench that provides the stimulus for the schematic. Following is the template for the configuration information to add to the bottom of the testbench:

```
CONFIGURATION cfg_name OF testbench_entity IS

   FOR testbench_arch
      for all : instantiated_comp use entity
```

```
work.entity(architecture);
         for architecture
            for all : core_name use entity
XilinxCoreLib.C_DECODE_BINARY_V1_0(behavioral)
                        <<configuration information
   provided in the CORE instantiation template
   core_name.vho>>
            end for;
         end for;
      end for;
   END FOR;
END TOP_cfg;
```

Verilog Simulation

CORE Generator Verilog simulation models are linked to the design using “include” statements. When a CORE is created, an instantiation template, *core_name*.veo, is generated in the project directory that contains the necessary information. For successful simulation in ISE projects, this information must be included in the CORE Generator symbol.

For schematic designs in Foundation Series ISE projects that contain CORE Generator created macros, you must use the following procedure to incorporate the necessary information in the CORE Generator symbol before using the symbol in a schematic:

1. Rename the *core_name*.veo file to ***core_name*.v**.
2. Add the *core_name*.v file to the project.
3. In the Source window, click on the ***core_name*.v** file.
4. Double click **Create Schematic Symbol** in the Process window.
5. When asked if you should overwrite the existing symbol, click **Yes**.
6. Add the symbol to the schematic.

Chapter 10

HDL Library Mapping

Foundation Series ISE includes an HDL library mapping feature that provides a method to include HDL files as part of a library. This chapter contains the following sections.

- “Design Sources and Libraries”
- “Project Navigator Source Libraries”
- “Named VHDL Libraries”
- “Moving Files to a Library”
- “Removing Files from a Library”

Design Sources and Libraries

Libraries allow sharing of design elements between sources in a design and between designs. This makes designs easier to manage and allows for design reuse. The HDL library mapping feature provides a method for you to tell the Project Navigator which files are library files, and depending on the language, what 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 for 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" would allow it to access anything that is 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.

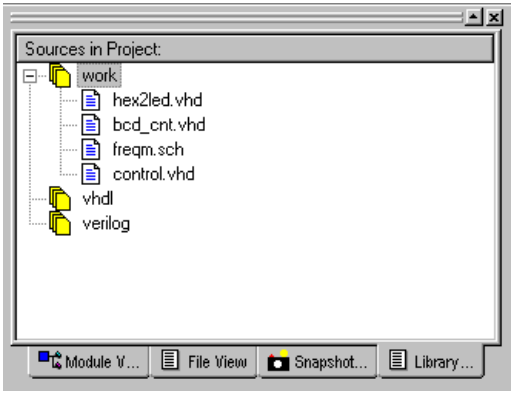
Verilog

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 isn't resolved in the set of input files, it will look in the specified library files and/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 only 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 will only compile and use "foomod" and ignore the rest of the modules.

Project Navigator Source Libraries

In Foundation Series ISE, the default library where the current design sources are placed is called "work." Depending on the synthesis tool for the project, a VHDL and / or Verilog library is also provided. The libraries and elements included in them are listed in the Library View of the Source window. An example Library View is shown in the following figure.



The “work” library contains all of the design sources in the project. It corresponds to the sources in the Module View. This 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).

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 the “Named VHDL Libraries” section and add any number of files to those libraries.

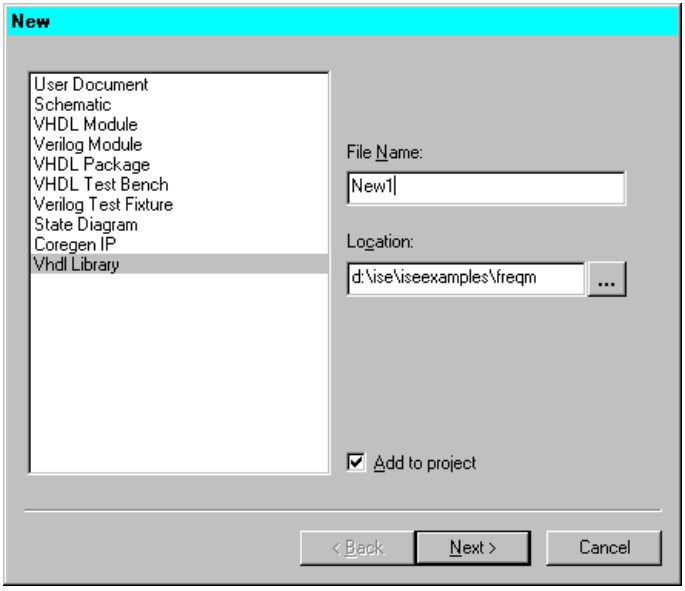
If you are using FPGA Express or XST Verilog, the “verilog” library is included in the Library View. The Verilog library contains Verilog files. This library cannot be deleted.

The Add, Add Copy, removing files from this library is the same as adding or removing them from the project.

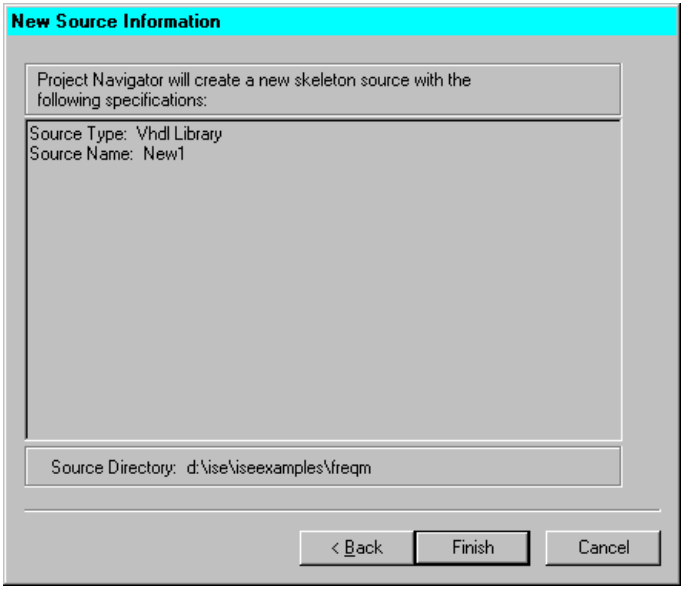
Named VHDL Libraries

A file cannot be added directly to the "vhdl" library displayed in the Library View. You must first create a named directory to hold the file. Use the following procedure to create named VHDL directories.

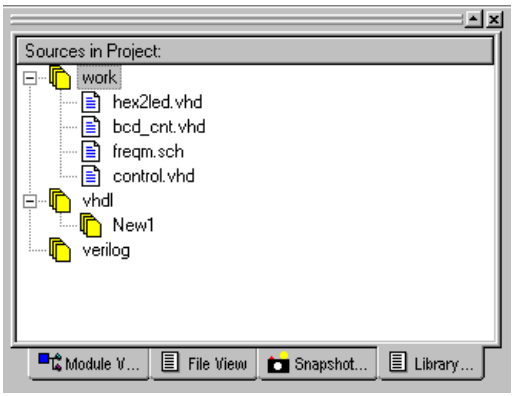
1. Open or create your project (see the "Creating a Project" chapter).
2. Select **`Project`** → **`New Source`** from the Project Navigator to access the New source window.
3. Select **`Vhdl Library`** from the list of available source types displayed in the New source window. The Vhdl Library selection is only available 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 window to proceed.



7. Select the Library View tab on the Source window. As shown in the following figure, the newly created VHDL library appears under “vhdl.”



8. To add a file to the library, the file must be first added to the project and then moved (see the “Moving Files to a Library” section) from the Work directory to the desired library.

Renaming VHDL Libraries

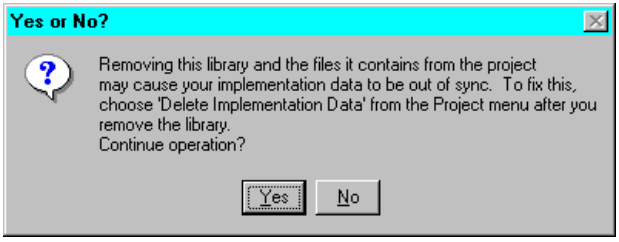
Use the following procedure to rename a VHDL library.

1. In the Library View of the Source window, click on the name of the VHDL library you want to rename.
2. Select **Source** → **Rename** from the Project Navigator menu.
3. Modify the library name as desired.

Removing VHDL Libraries

Use the following procedure to remove a VHDL library.

1. In the Library View of the Source window, click on the name of the VHDL library you want to remove.
2. Select **Source** → **Remove** from the Project Navigator menu.
3. A message box appears to remind 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. (The library is deleted. Its files are not deleted, just removed from the project.)



4. Select **Project** → **Delete Implementation Data** after you remove the library.

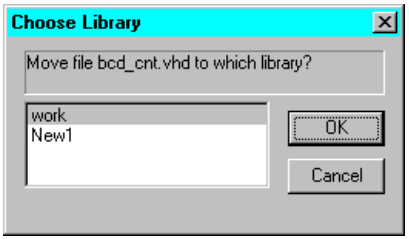
Note After a library has been deleted there is no way to bring it back except to recreate it.

Moving Files to a Library

Files that have been added to the project can be moved from one library to another. A file may not 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.

Use the following procedure to move files between libraries.

1. In the Library View tab, click on the name of the file you want to move.
2. Select **Source** → **Move to Library**.
3. Select the library from the Choose Library dialog box and then click **OK**.



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

Removing Files from a Library

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

Chapter 11

Design Constraints/UCF File

You can enter constraints to control the synthesis and implementation of your design. This chapter gives an overview of the various ways you can enter constraints on your design. It also provides information on the User Constraints File (UCF) and the types of constraints that can be entered in it. This chapter contains the following sections.

- "Setting Synthesis Constraints"
- "Setting Implementation Constraints"
- "Constraints Processing Overview"
- "Constraints File Overview"
- "ISE User Constraints File (UCF)"
- "The Xilinx Constraints Editor"
- "Timing Constraints"
- "Layout Constraints"
- "Efficient Use of Timespecs and Layout Constraints"
- "Standard Block Delay Symbols"
- "Table of Supported Constraints"
- "Basic UCF Syntax Examples"

Setting Synthesis Constraints

Before you start synthesis to optimize your design for the targeted device, you can optionally set performance constraints, attributes, and optimization controls. Entering your design requirements as constraints can improve the placement and routing results of your design. You can control the synthesizing of your design by setting

process properties in the Project Navigator or in the User Constraints File. Each synthesis tool also has additional methods for entering synthesis constraints.

XST Constraints

For projects that use the XST synthesis tool, you can set synthesis constraints in the following ways:

- Using the Project Navigator's Synthesis process properties described in the "Setting XST Synthesis Options" section of the "Synthesis" chapter.
- Entering XST-specific constraints and attributes directly into the HDL code. Refer the *XST User Guide* for information on XST-specific constraints.
- Entering constraints in a UCF file. XST cannot apply certain constraints from the UCF file. Constraints that cannot be applied are passed through to the implementation tools for processing. Refer to the *XST User Guide* for information on XST and UCF constraints.
- Using the Xilinx Constraints Editor to automate entry of constraints in the UCF file. Refer to the Xilinx *Constraints Editor Guide* for detailed information on using the Constraints Editor.
- Entering constraints in the XST constraints file. Refer to the *XST User Guide* for detailed information on the XST constraints file.

FPGA Express Constraints

For projects that use FPGA Express as the synthesis tool, you can set synthesis constraints in the following ways:

- Using the Project Navigator's Synthesis process properties described in the "Setting FPGA Express Synthesis Options" section of the "Synthesis" chapter.
- Using the Express Constraints Editor. There is an overview of the Express Constraints Editor in the "Detailed Information on XST" section of the "Synthesis" chapter. For detailed information on the Express Constraints Editor refer to the FPGA Express online help.

- Entering constraints in a UCF file. FPGA Express cannot apply certain constraints from the UCF file. Constraints that cannot be applied are passed through to the implementation tools for processing. Refer to the *XSI Guide* for information on FPGA Express and UCF constraints.
- Using the Xilinx Constraints Editor to automate entry of constraints in the UCF file. Refer to the Xilinx *Constraints Editor Guide* for detailed information on using the Constraints Editor.

Setting Implementation Constraints

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. You can set multiple properties to control the implementation processes for the design. For CPLDs, they control how a design is translated and fit.

You set implementation constraints in the following ways.

- Using the Project Navigator's implementation Process Properties described in the "FPGA Implementation Options" section of the "Implementing the Design" chapter and the "CPLD Implementation Options" section of the "Implementing the Design" chapter.
- Entering constraints in a UCF file as described in the "ISE User Constraints File (UCF)" section.
- Using the Xilinx Constraints Editor to automate entry of constraints in the UCF file. Refer to the Xilinx *Constraints Editor Guide* for detailed information on using the Constraints Editor.
- Entering attributes directly on a schematic. Refer to the ECS Schematic Editor online help for detailed information on this option.
- Entering constraints directly in HDL design code. Refer to the *XST User Guide* and *XSI Guide* for detailed information on this method.
- You can also enter design constraints in the Floorplanner. Refer to the *Floorplanner Guide* for details.

Constraints Processing Overview

You control the synthesis and/or implementation of a design by entering constraints. There are two basic types of constraints that you can apply to a design: location constraints and timing constraints.

Location constraints control the mapping and positioning of the logic elements in the target device. The most common location constraints are pad constraints. They are used to lock the pins of the design to specific I/O locations so that the pin placement is consistent from revision to revision.

Timing constraints tell the software which paths are critical, and therefore, need closer placement and faster routing. Conversely, timing constraints also tell the software which paths are not critical and, therefore, do not need closer placement or faster routing. Both the placer and the router can be timing constraint driven.

For a complete listing of all supported constraints, refer to the *Libraries Guide* (Chapter 12, "Attributes, Constraints, and Carry Logic,"). For a more complete discussion of how timing constraints work in Foundation, refer to the *Development System Reference Guide* ("Using Timing Constraints,"). For information on all attributes, including timing constraints, used in CPLD designs, refer to the Foundation Series ISE online help.

Constraint Entry Mechanisms

With the Xilinx implementation tools, you control the implementation of a design by defining constraints that affect the mapping and layout of the physical circuit. Additionally, you can specify the "path" timing requirements of the circuit to obtain the best results and allow the implementation tools to choose the layout which best satisfies these requirements.

The various design constraints available within ISE can be entered at the time you create the design (i.e., the logical domain) or after the design is mapped (that is, the physical domain).

Constraints entered in the logical domain are created in the following ways

- Entered into the schematic

- Applied to a synthesis process and then forward-annotated through a netlist constraints file (NCF)
- Created with the Constraints Editor (see "Case Sensitivity" section)

Constraints entered in the physical domain are entered directly into the Physical Constraints File (PCF). These constraints are conceptually the same as those entered during design creation; however, they are directly related to objects within the physical design database and are therefore applied using the PCF syntax.

The following figure illustrates the constraints entry approach for the Xilinx implementation tools.

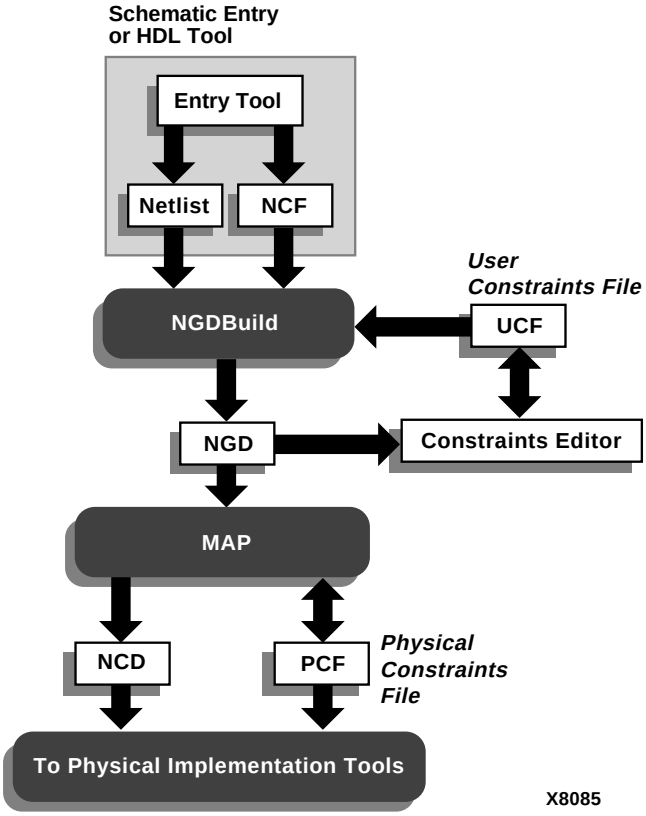


Figure 11-1 Constraint Entry Flow

Translating and Merging Logical Designs

The process of implementing a design starts with a logical design file (NGD) that represents the design created by the Translate process (NGDBuild) as shown in the Figure 11-1.

The NGD file contains all of the design's logic structures (gates) and constraints. The NGD file is produced through the NGDBuild process which controls the translation and merging of all of the related logic design files.

All design files are translated from industry standard netlists into intermediate NGO files by a netlist translation program EDIF2NGD. The exception to this rule is logic, which is created through the use of LogiBLOX components. LogiBLOX components may be compiled directly in memory, and are, therefore never written to disk as a separate intermediate NGO file.

Constraints File Overview

The following subsections describe the Netlist Constraints File, User Constraints File, and the Physical Constraints File.

Netlist Constraints File (NCF)

The Netlist Constraints File (NCF) is an ASCII file generated by the synthesis program. It contains the logical constraints entered in the design.

User Constraints File (UCF)

The User Constraint File was developed to provide a convenient mechanism for constraining a logical design without returning to the design entry tools. UCF constraints intentionally overwrite constraints that are present in the netlist.

UCF constraints override any constraints contained within the netlist created by the schematic or synthesis tools. A constraint that is being applied via the UCF file must specify the complete hierarchical path name for the instance or net being constrained.

UCF constraints are considered more significant because they appear later in the design flow and provide a mechanism for establishing or

modifying logical design constraints without requiring you to re-enter a schematic or synthesis tool.

The process of building the complete logical design representation (NGD files) is the job of NGDBuild. In developing this complete design database, NGDBuild annotates design constraints with those it finds in a UCF file. If a UCF file exists with the same name as the top-level netlist then it will automatically be read. Otherwise, you must indicate a specific file for User Constraints in the Options dialog box. The syntax for the UCF constraints file is explained (on a per-constraint basis) in the "Timing Constraints" section.

Physical Constraints File (PCF)

(FPGA only) The layout tools work on the physical design, so the PCF file is written in terms that these tools can readily interpret. Layout and timing constraints are written in terms of the physical design's components (COMPs), fractions of COMPs (BELs), and collections of COMPs (macros).

Because of this different design viewpoint, the PCF syntax is not necessarily the same as that used in the logical design constraint files (UCF / NCF). Furthermore, because the PCF file is written for the physical design implementation tools, its syntax may not be as intuitive as the UCF file. Regardless of the syntactical challenges associated with using a PCF file, many designers will choose to work at the physical level of design abstraction for the following reasons.

- It is readily modified and immediately applicable to the present task —implementing an FPGA (that is, there is no need to re-run NGDBuild or MAP in order to run layout or analysis tools).
- The implications of logical design structures on the physical design's implementation only become obvious once the design is evaluated using the physical tools. Altering the PCF file for "what-if" analysis can be desirable.
- Certain constraints are only available within the PCF file.

Note If you modify the PCF file, you should be certain that you enter your constraints after the line "SCHEMATIC END ;". Otherwise, your constraints will be overwritten every time MAP is re-executed.

Case Sensitivity

Since EDIF is a case-sensitive format, the Foundation constraints are case sensitive as well. Always specify the net names and instance names exactly as they are in your schematic or code. Be consistent when using TNMs and other user-defined names in your constraints file; always use the same case throughout. For site names (such as "CLB_R2C8" or "P2"), you should use only upper case letters, since site names within Xilinx devices are all upper case.

ISE User Constraints File (UCF)

The user constraints file (.ucf) is an ASCII file that holds timing and location constraints. It is read by NGDBuild during the translate process and is combined with an EDIF netlist into an NGD file (see the "Implementing the Design" chapter). By default, each ISE project has a UCF file with the same name as the top-level netlist. The default UCF file is created automatically when the project is created.

You can access the UCF file for a project by double-clicking on **Edit UCF File** in the Process section of the Process window. The UCF file is opened in Notepad (or your usual text editor). Or, you can click **Constraints Editor** to access a GUI to automate constraint entry into the UCF file. You can change the UCF file to use for the project using implementation Process Properties (see "FPGA Implementation Options" section or the "CPLD Implementation Options" section of the "Implementing the Design" chapter for details).

Note Whenever you modify a UCF file, you must rerun the implementation processes for the changes to be reflected. A Notice dialog box appears when you save the file. You can choose to Reset the implementation processing or Retain the current implementation results.

Two examples are included here to introduce you to UCF files.

The following example shows how to lock I/Os to pin locations and how to write Timespec and Timegroup constraints in the UCF.

Note You can also lock pin locations within the Project Navigator by selecting the Backannotate Pin Locs (FPGAs) or Lock Pin (CPLDs) process. See the "FPGA Implementation Flow" section of the "Implementing the Design" chapter for details.

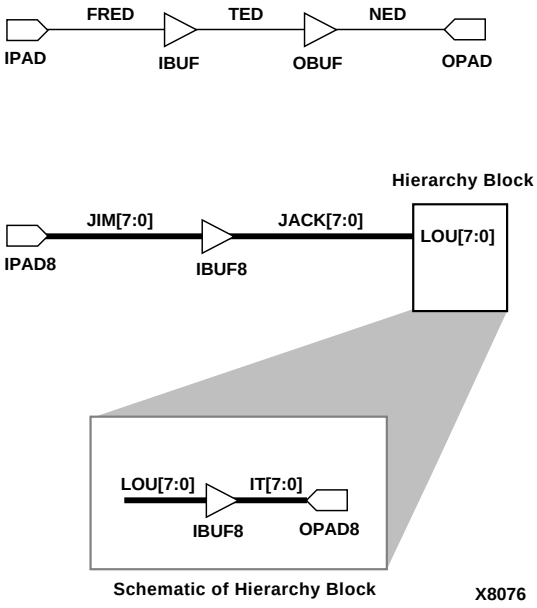


Figure 11-2 Locking I/Os to Pin Locations

```
# This is a UCF comment

# The constraints below lock the I/O signals to
 pads.

# The net name that connects to the pad is used to
# constrain the I/O.

# The pin grid array packages use pin names like
 B3 or
# T1, instead of P<Pin Number>.
```

```
# Lock the input pins

NET FRED LOC = P18;
NET JIM<0> LOC = P20;
NET JIM<1> LOC = P23;
NET JIM<2> LOC = P24;
NET JIM<3> LOC = P25;
NET JIM<4> LOC = P26;
NET JIM<5> LOC = P27;
NET JIM<6> LOC = P28;
NET JIM<7> LOC = P38;

# Lock the output pins

NET NED LOC = P19;
NET HIERARCHY_BLOCK/<IT0> LOC = P44
NET HIERARCHY_BLOCK/<IT1> LOC = P45
NET HIERARCHY_BLOCK/<IT2> LOC = P46
NET HIERARCHY_BLOCK/<IT3> LOC = P47
NET HIERARCHY_BLOCK/<IT4> LOC = P48
NET HIERARCHY_BLOCK/<IT5> LOC = P49
NET HIERARCHY_BLOCK/<IT6> LOC = P50
NET HIERARCHY_BLOCK/<IT7> LOC = P462
```

For more information on constraint precedence, refer to the “Using Timing Constraints” chapter in the *Development System Reference Guide*.

The following example shows how to specify timing constraints.

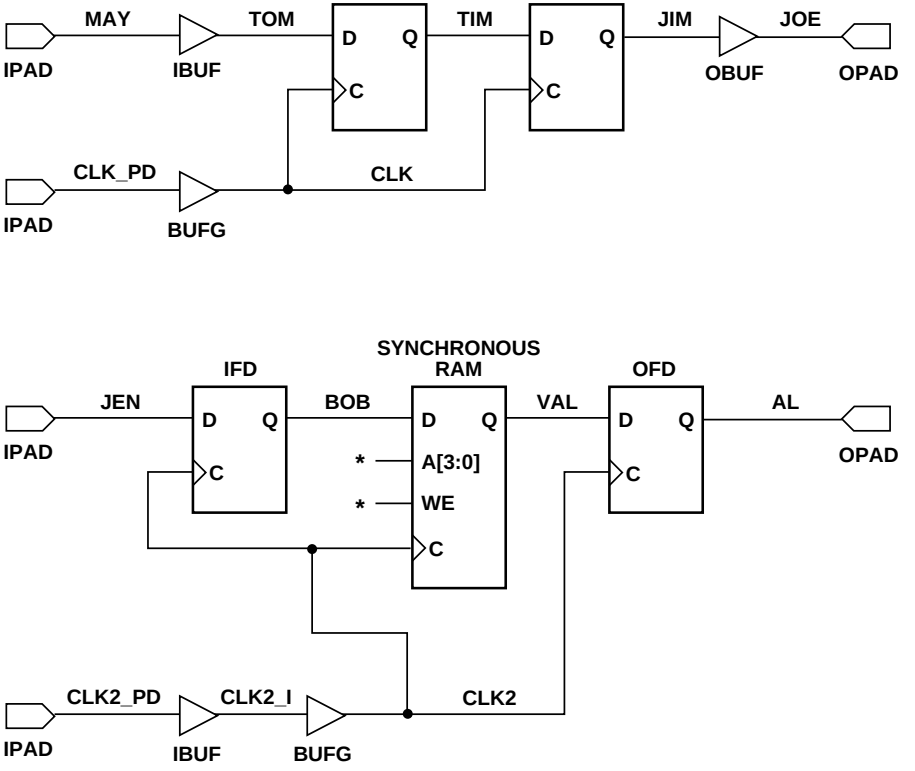


Figure 11-3 Specifying Timing Constraints

```
---User Constraint File (UCF):

# This is a comment

# Period specifies minimum PERIOD of CLK net. Offset specifies that
# data on MAY can arrive up to 6 ns before the clock edge arrives
on CLK.

# NOTE: Period constraints do not apply to elements in input or
output pads.
```

```
NET CLK PERIOD = 20 ns ;
NET MAY OFFSET = IN 6ns before CLK_PD ;

# Groups all clocked loads of CLK2 into CLK2_LOADS timegroup
# Groups all clocked loads of VAL into VAL_LOADS
# timegroup TNM # => Timegroup NaMe

NET CLK2 TNM=CLK2_LOADS ;
NET VAL TNM=VAL_LOAD ;

# Specifies worst case speed of path from IPAD to CLK2 # loads.
Includes
# pad, buffer, and net delays. TS0l is a Timespec identifier; it can
# have names of the form TS<string>. PADS (CLK2_PD) is a Timegroup
name
# specified inside of a Timespec.

TIMESPEC TS01=FROM PADS (CLK2_PD) TO CLK2_LOADS=15ns ;

# Specifies the maximum frequency for all loads clocked by CLK2.

TIMESPEC TS02=FROM CLK2_LOADS TO CLK2_LOADS=30Mhz;

# Specifies the minimum delay on the path from Synchronous RAM to
OFD.
# Includes clock-to-out delay, net delay, and setup time.

TIMESPEC TS03=FROM CLK2_LOADS TO VAL_LOAD=15000ps ;
```

The Xilinx Constraints Editor

The Xilinx Constraints Editor is a Graphical User Interface (GUI) that provides a convenient way for you to create certain new constraints. Constraints created with the Constraints Editor are written to the UCF (User Constraints File). See the “Constraints File Overview” section.

For more information on the Constraints Editor, see the *Constraints Editor Guide*, an online book.

Timing Constraints

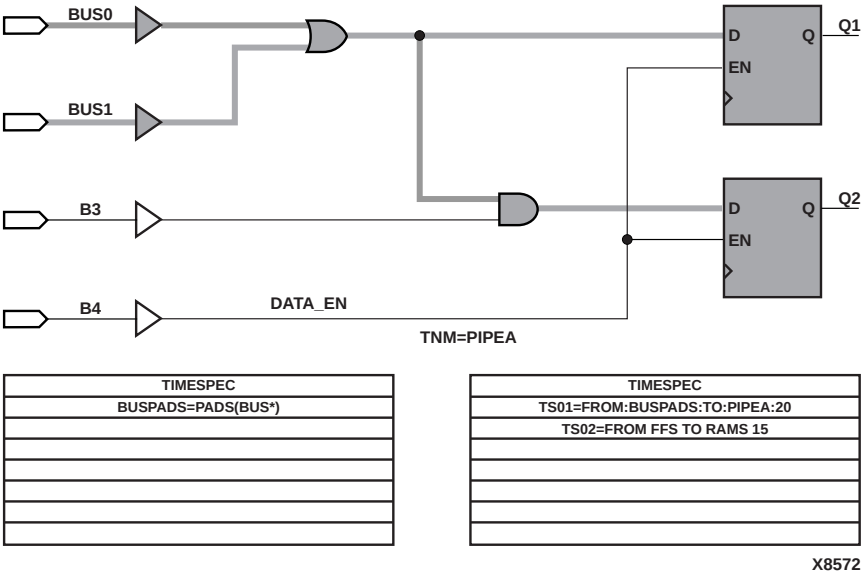
The following subsections discuss timing constraints. Many timing constraints can be created using the Constraints Editor. If a constraint can be created with the Constraints Editor, it will be noted in the sections that follow.

The “From:To” Style Timespec

When using the From:To style of constraint, the path(s) that are constrained are specified by declaring the start point and end point, which must be a pad, flip-flop, latch, RAM, or user-specified sync point (see TPSYNC). To group a set of endpoints together, you may attach a TNM attribute to the object (or to a net that is an input to the object). With a macro, the TNM traverses the hierarchy to tag all relevant objects. A TIMEGRP is a method for combining two or more sets of TNMs or other TIMEGRPs together, or alternatively, to create a new group by pattern matching (grouping a set of objects that all have output nets that begin with a given string)

You can create a From:To timespec with the Constraints Editor.

You use TNMs to identify a group of design objects which are to be referenced within a Timespec. If a TNM is placed on a net, the Foundation tools determine TNM membership by tracing forward from the specified net to all the valid endpoints of the net. Refer to the *Development System Reference Guide* (“Using Timing Constraints”) for more information on this subject. The following schematic shows an example of TNM, TIMESPEC, and TIMEGRP statements.



The following file corresponds to the preceding figure.

```
# This is a comment line
# UCF FROM:TO style Timespecs

  NET DATA_EN TNM = PIPEA ;
  TIMEGRP BUSPADS = PADS(BUS*) ;
  TIMESPEC TS01 = FROM:BUSPADS:TO:PIPEA:20 ;

# Spaces or colons (:) may be used as field
   separators

  TIMESPEC TS02 = FROM FFS TO RAMS 15 ;
```

The first line of the above example illustrates the application of the TNM (Timing Name) PIPEA to the net named DATA_EN. The second line illustrates the TIMEGRP design object formed using a pattern matching mechanism in conjunction with the predefined TIMEGRP "PADS". In this example, the TIMEGRP named BUSPADS will include only those PADs with names that start with *BUS*.

Each of the user-defined Timegroups is then used to define the object space constrained by the timing specification (Timespec) named TS01. This timing specification states that all paths from each member of the BUSPADS group to each member of the PIPEA group need to have a path delay that does not exceed 20 nanoseconds (ns are the default units for time). The TIMESPEC TS02 constraint illustrates a similar type of timing constraint using the predefined groups FFS and RAMS.

Note All From:To Timespecs must be relative to a Timegroup. The above example illustrates that you can define Timegroups either explicitly (TIMEGRPs) or implicitly (TNMs), or they may be predefined groups (PADS, LATCHES, FFS, RAMS).

There is an additional keyword that you can add to the From:To specification that allows the user to narrow the set of paths that are covered—THRU. By using the From:Thru:To form of a Timespec, you are able to constrain only those paths that go through a certain set of nets, defined by the TPTHRU keyword, as shown in the following example.

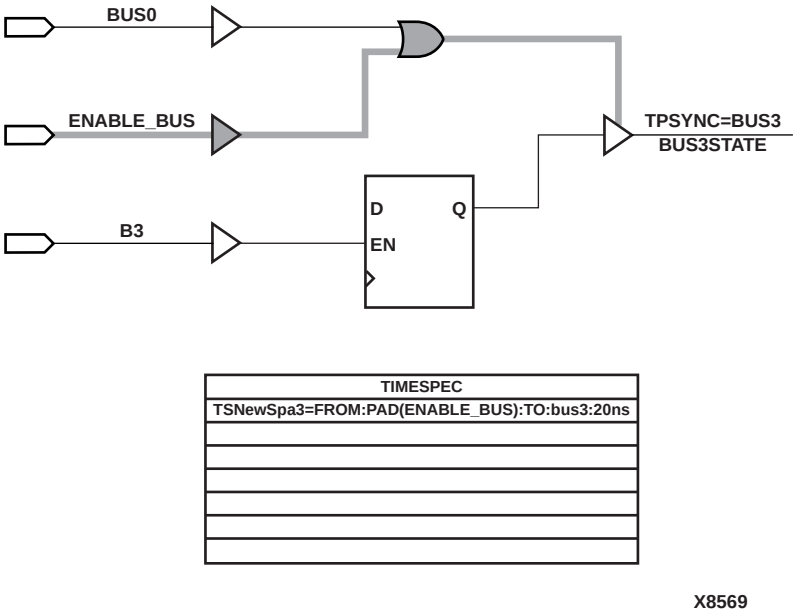
```
# UCF example of FROM:TO Timespec using THRU

  NET $1I6/thisnet TPTHRU=these ;
  NET $1I6/thatnet TPTHRU=these ;

  TIMEGRP sflops=FFS(DATA*) ;
  TIMEGRP dflops=FFS(OUTREG*) ;

  TIMESPEC TS23=FROM:sflops:THRU:these:TO:dflops:20
   ;
```

Here, only those paths that go from the Q pin of the *sflops* through the nets `$1I6/thisnet` and `$1I6/thatnet` and on to the D pin of *dflops* will be controlled by TS23.

Using TPSYNC

(FPGA only.) You can define any node as a source or destination for a Timespec with the TPSYNC keyword. The use of TPSYNC is similar to TPTHRU—it is a label that is attached to a set of nets, pins, or instances in the design.

For example, suppose a design has a PAD ENABLE_BUS that must arrive at the enable pin of several different 3-state buffers in less than a specified time. You can define that 3-state buffer as an endpoint for a timing spec. The following figure illustrates TPSYNC.



The following UCF file corresponds to the above example.

```
# TPSYNC example; pad to a 3-state buffer enable
 pin
# Note TPSYNC attached to 3-state buffer's output
 NET

NET BUS3STATE TPSYNC=bus3;
TIMESPEC
 TSNewSpc3=FROM:PAD(ENABLE_BUS):TO:bus3:20ns;
```

In the NET statement shown above, the TPSYNC is attached to the output net of a 3-state buffer called *BUS3STATE*. If a TPSYNC is attached to a net, then the source of the net is considered to be the endpoint (in this case, the 3-state buffer itself). The subsequent

TIMESPEC statement can use the TPSYNC name just as it uses a TNM name.

The next TPSYNC UCF file example shows you how to use the keyword PIN instead of NET if you want to attach an attribute to a pin.

```
# Note TPSYNC attached to 3-state buffer's enable
 PIN

PIN $1I6/BUSMACRO1/TRIBUF34.T TPSYNC=bus1;
TIMESPEC
 TSNewSpc1=FROM:PAD(ENABLE_BUS):TO:bus1:20ns;
```

In this example, the instance name of the 3-state buffer is stated followed by the pin name of the enable (.T). If a TPSYNC is attached to a primitive input pin, then the primitive's input is considered the startpoint or endpoint for a timing specification. If it is attached to a output pin, then the output of the primitive is used.

The last TPSYNC example shows you how to use the keyword INST if you want to attach an attribute to a instance:

```
# Note TPSYNC attached to 3-state buffer INSTANCE
 (UCF
# file)

INST $1I6/BUSMACRO2/BUFFER_2 TPSYNC=bus2;
TIMESPEC
 TSNewSpc2=FROM:PAD(ENABLE_BUS):TO:bus2:20ns;
```

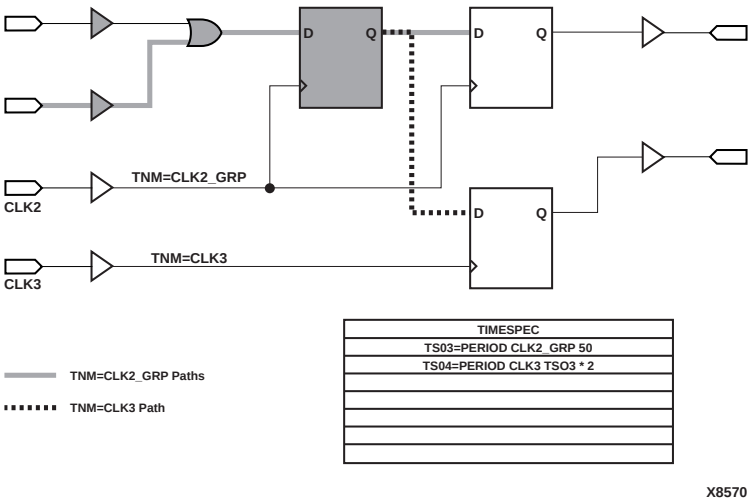
If a TPSYNC is attached to an instance, then the output of the instance is considered the startpoint or endpoint for a timing specification.

The Period Style Timespec

The TIMESPEC form of the PERIOD constraint allows flexibility in group definitions and allows you to define clock timing relative to another TIMESPEC.

You can create a Period constraint with the Constraints Editor.

The following schematic example illustrates the use of the PERIOD Timespec referenced to timegroups *CLK2_GRP* and *CLK3.*



The following syntax is the corresponding UCF file.

```
# UCF PERIOD style Timespecs

  NET CLK2 TNM = CLK2_GRP ;
  NET CLK3 TNM = CLK3 ;

  TIMESPEC TS03 = PERIOD CLK2_GRP 50 ;
  TIMESPEC TS04 = PERIOD CLK3 TS03 * 2 ;
```

Furthermore, the example shows how constraints and nets may be given the same name because they occupy separate name-spaces. Also, it shows the constraint syntax whereby one Timespec is defined relative to another (the value of TS04 is declared to be two times that of TS03).

The PERIOD constraint covers all timing paths which start or end at a register, latch, or synchronous RAM that is clocked by the referenced net. The only exception to this rule are paths to output pads, which are not covered by the PERIOD constraint. (Input pads, which are the source of a “pad-to-setup” timing path for one of the specified synchronous elements, *are* covered by the PERIOD constraint.)

The flexibility of the TIMESPEC form of the PERIOD constraint arises from being able to modify the contents of the TIMEGRP once the design has been mapped. By adding or removing objects from the TIMEGRP, which are listed in the PCF file, you can alter the paths that are covered by the PERIOD constraint.

If you do not need the flexibility offered by the TIMESPEC form, you can use the NET form of the PERIOD constraint may be used. The syntax for the NET form of the PERIOD constraint is simpler than the TIMESPEC form, while continuing to provide the same path coverage. The following example illustrates the syntax of the NET form of the PERIOD constraint.

```
# NET form of the PERIOD timing constraint
# (no TSidentifier)

  NET CLK PERIOD = 40 ;
```

This is the recommendation of using PERIOD on a single clock design in which data does not pass between the clock domains.

With the Foundation 1.5 release, PERIOD will now include clock skew in the path analysis.

The Offset Constraint

Use offsets to define the timing relationship between an external clock and its associated data-in or data-out-pin. Using this option, you can do the following.

- Calculate whether a setup time is being violated at a flip-flop whose data and clock inputs are derived from external nets.
- Specify the delay of an external output net derived from the Q output of an internal flip-flop being clocked from an external device pin.

You can create a Pad to Setup or Clock to Pad offset constraint with the Constraints Editor.

There are basically three types of offset specifications.

- Global
- Specific

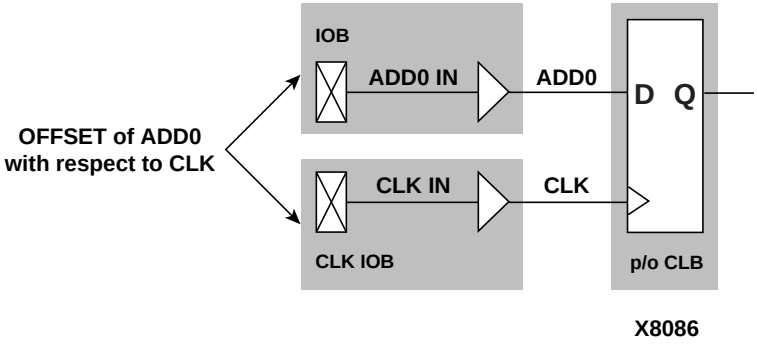
- Group

Since the global and group OFFSET constraints are not associated with a single data net or component, these two types can also be entered on a TIMESPEC symbol in the design netlist with Ts*id.* See the "Using Timing Constraints" chapter in the *Development System Reference Guide* for details.

In the following example, the OFFSET constraint is applied to a net that connects with a PAD (as shown in the figure later in this section). It defines the delay of a signal relative to a clock and is only valid for registered data paths. The OFFSET constraint specifies the signal delay external to the chip, allowing the implementation tools to automatically adjust relevant internal delays (CLK buffer and distribution delays) to accommodate the external delay specified with the following.

```
# Net form of the OFFSET timing constraint

  NET ADD0_IN OFFSET = IN 14 AFTER CLK ;
```

In analyzing OFFSET paths, the Xilinx timing tools adjust the PERIOD associated with the constrained synchronous element based on both the timing specified in the OFFSET constraint and the delay of the referenced clock signal. In the following figure, assume a delay of 8ns for the signal CLK to arrive at the CLB, a 5ns setup time for ADD0, and a 14ns OFFSET delay for the signal ADD0. Assume a period of 40ns is specified. The Foundation tools allocate 29ns for the signal ADD0 to arrive at the CLB input pin (40ns - 14ns + 8ns - 5ns = 29ns).



This same timing constraint could be applied using the FROM:PADS:TO:FFS timing constraint. However, using a From:To methodology would require you to know the intrinsic CLK net delay, and you would have to adjust the value assigned to the From:To Timespec. The internal CLK net delay is implicit in the OFFSET / PERIOD constraint. Furthermore, migrating the design to another speed grade or device would require modification of the From:To Timespec to accommodate the new intrinsic CLK net delay. An alternative solution is to use the flip-flop in the IOB of certain FPGA architectures (XC4000E / EX, for instance), as the clock-to-setup time *is* specified in the *Programmable Logic Data Book*.

Note Relative Timespecs can only be applied to similar Timespecs. For example, a PERIOD Timespec may be defined in terms of another PERIOD Timespec, but not a FROM:TO Timespec.

Ignoring Paths

(FPGA only.) When you declare a a Timespec that includes paths where the timing is not important, the tools may create a less optimal route since there is more competition for routing resources. This problem can be alleviated by using a TIG (timing ignore) attribute on the non-critical nets. TIG causes all paths that fan out from the net or pin where it is applied to be "ignored" during timing simulation.

You can create a Timing Ignore constraint with the Constraints Editor.

The following syntax indicates that `$1I456/slow_net` should not have the Timespec TS01 or TS04 applied to it.

```
#Timespec-specific TIG example (UCF file)

  NET $1I456/slow_net TIG=TS01, TS04 ;
```

On the other hand, the following syntax indicates that the layout tools should ignore paths through the `$1I456/slow_net` net for *all* known Timespecs.

```
#Global TIG example (UCF file)

  NET $1I456/slow_net TIG
```

Controlling Skew

(FPGA only.) Skew is the difference between the minimum and maximum of the maximum load delays on a net. You can control the maximum allowable skew on a net by attaching the MAXSKEW attribute directly to the net.

```
#MAXSKEW example (UCF file)

  NET $1I345/net_a MAXSKEW=3 ;
```

The above example indicates that 3 ns is the maximum skew allowed on $1I345/net_a. For a detailed example of how MAXSKEW works, see the "Additional Timing Constraints" section in the *Development System Reference Guide*.

Constraint Precedence

A design may assign a precedence to Timespecs only within a certain class of constraints. For example, you may specify a priority for a particular From:To specification to be greater than another, but you may not specify a From:To constraint to have priority over a TIG constraint. The following example illustrates the explicit assignment of priorities between two same-class timing constraints, the lowest number having the highest priority.

```
# Priority UCF example

  TIMESPEC TS01 = FROM GROUPA TO GROUPB 40
PRIORITY 4;
  TIMESPEC TS02 = FROM GROUP1 TO GROUP2 35
PRIORITY 2;
```

The following sections illustrate the order of precedence for the various types (and various sources) of timing constraints.

Across Constraint Sources

Across constraint sources, the following priorities apply

- Physical Constraint File (PCF)—the highest priority
- User Constraint File (UCF)
- Input Netlist / Netlist Constraint File (NCF)—the lowest priority

Within Constraint Sources

Within constraint sources, the following priorities apply.

- TIG (Timing Ignore)—the highest priority
- FROM:*source*:THRU:*point*: TO:*destination* specification

  The priority of each type of FROM:THRU:TO specification is as follows (highest priority is listed first).

  FROM:USER1:THRU:USER_T:TO:USER2 specification(USER1 and USER2 are user-defined groups)

  FROM:USER1:THRU:USER_T:TO:FFS specification
  or
  FROM:FFS:THRU:USER_T:TO:USER2 specification (FFS is any pre-defined group)

  FROM:FFS:THRU:USER_T:TO:FFS specification
- FROM:*source*:TO:*destination* specification

  The priority of each type of FROM:TO specification is as follows (highest priority is listed first).

  FROM:USER1:TO:USER2 specification

  FROM:USER1:TO:FFS specification
  or
  FROM:FFS:TO:USER2 specification

  FROM:FFS:TO:FFS specification
- PERIOD specification
- “Allpaths” type constraints—the lowest priority

Layout constraints also have an inherent precedence which is based on the type of constraint and the site description provided to the tools. If two constraints have the same priority and cover the same path, then the last constraint in the constraint file will override any other constraints that overlap.

Layout Constraints

(FPGA only.) The mapping constraints in the example below illustrate some of the capabilities to control the implementation process for a design. The OPTIMIZE attribute is attached to the block of logic associated with the instance "GLUE."All of the combinatorial logic within the block GLUE will be optimized for speed (minimizing levels of logic) while other aspects of the design will be processed by the default mapping algorithms (assuming the design-based optimization switches are not issued).

```
# Mapping constraint

  INST GLUE OPTIMIZE = SPEED ;

# Layout constraint

  NET IOBLOCK/DATA0_IN LOC = P12 ;
```

The layout constraint in the example above illustrates the use of a full hierarchical path name for the net named DATA0_IN in the application of the I/O location constraint. In this example, IOBLOCK is a hierarchical boundary that contains the net DATA0_IN. Location constraints applied to "pad nets" are used to constrain the location of the PAD itself, in this case to site P12.

Note If the design contains a PAD, the constraint could have been just as easily applied to it directly (some design flows do not provide explicit I/O pads in the design netlist).

Converting a Logical Design to a Physical Design

The process of mapping translates a design from the logical design domain to the physical design domain. The MAP process creates both the physical design components (CLBs, IOBs, and so forth) and the physical design constraints (layout and timing). The physical design components are written into a Native Circuit Description (NCD) file. The physical design constraints are written into a Physical Constraints File (PCF).

As the design flow of the Figure 11-1 shows, MAP not only writes a PCF file, but also reads a specified pre-existing PCF file. MAP reads an existing PCF file in order to facilitate the overriding of constraints

that are contained within another logic design using the “last one wins” resolution mechanism provided by the PCF file. The following subsection briefly describes this approach.

“Last One Wins” Resolution

MAP creates new physical design constraints each time it converts a logical design into a physical design. The constraints that are created during this process are written into the “Schematic” section of the PCF file. This section is recreated each time MAP is run based on the constraints that are contained within the NGD file. The schematic section is always written at the top of the PCF file, and constraints that are in the PCF file but outside of the Schematic section (after the line “SCHEMATIC END”) are considered to be in the “User” section of the PCF file. The user section is read, syntactically checked, and rewritten each time MAP is run. Since these constraints always follow those written into the schematic section, they will always take precedence (following the “last-one-wins” rule).

Note If the design contains a PAD, the constraint could have been just as easily applied to it directly (some design flows do not provide explicit I/O pads in the design netlist).

Efficient Use of Timespecs and Layout Constraints

The previous section described the mechanisms available for constraining a design’s timing within the Foundation tools. The sections that follow summarize each of the constraints that are available.

The robust nature of the language enables you to define your design requirements at the highest level of abstraction first, and then fine tune the timing requirements by using more specific Timespecs, if needed. This is the methodology that will best describe your requirements to the tools.

The following observations help to illustrate the reasons why this methodology should be followed (from a tool runtime perspective).

- Using explicit Timegroups causes slower runtimes than using implicit timegroups arising from the use of constraints such as PERIOD.

- Processing larger Timegroups takes longer than processing smaller Timegroups.
- Using many specific Timespecs results in slower runtimes than using a smaller set of more general Timespecs.

In conclusion, overall design runtime is improved when a "qualified global" timing methodology is employed instead of a "thorough-detailed" timing methodology.

The "Starter Set" of Timing Constraints

The following examples clearly identify the "preferred" mechanism for controlling the timing of your design. The preferred method assumes a goal of getting the required results in the fastest run time possible. If the design has a single clock and required I/O timing that equals the clock period, all that you need are the three constraints shown in the following example.

```
# Global UCF example

  NET CLK1 PERIOD = 40 ;
  NET OUT* OFFSET = OUT 13 AFTER CLK ;

  TIMESPEC TS01 = FROM PADS TO PADS 40 ;
```

Note When you use net name wild cards in OFFSETS, make sure that the name is unique to valid nets; otherwise processing errors will occur.

If you need to account for extra delay external to the FPGA, then you could add the following.

```
NET INPUT* OFFSET = IN 8 BEFORE CLK ;
```

The PERIOD constraint covers all pad-to-setup and clock-to-setup timing paths. The OFFSET constraint covers the clock-to-pad timing for each of the output nets beginning with *OUT*. Both the OFFSET and PERIOD constraints account for the delay of the Clock Buffer/Net in the I/O timing calculations.

The following PCF fragment illustrates the differences in syntax between the UCF and PCF languages. In addition to the syntactical changes, remember that net and instance names may change. As an example, one of the net matches resulting from the UCF "NET *OUT**" constraint is now applied to "COMP *OUT1_PAD*". The name

OUT1_PAD is the name assigned to the pad instance. In addition to name changes, another difference is the verbosity of the PCF. In the PCF there is additional syntax for “MAXDELAY,” “TIMEGRP,” and “PRIORITY.” These are all optional qualifications of the Timespec within the UCF, but written explicitly to the PCF file illustrating the full flexibility of the language.

```
# Global PCF example
SCHEMATIC START;
. . .
NET PERIOD “CLK_IN” = 40 nS HIGH 50.00% ;
COMP “OUT1_PAD” OFFSET = OUT 40 ns AFTER COMP
 “CLK”;
COMP “OUT2_PAD” OFFSET = OUT 40 ns AFTER COMP
“CLK”;COMP “INPUT1_PAD” OFFSET = IN 28 ns BEFORE
 COMP
“CLK”;
TS01 = MAXDELAY FROM TIMEGRP “PADS” TO TIMEGRP
 “PADS”
      40000 pS PRIORITY 0;

SCHEMATIC END;
```

The next UCF example illustrates the use of both global constraints (PERIOD, OFFSET) to generally constrain the design and detailed Timespecs (FROM:THRU:TO) to provide fast and slow exceptions to the general timing requirements. Because the amount of constraints placed on a design directly impact runtime, Xilinx recommends that you first apply global constraints, then apply individual constraints only to those elements of the design that require additional constraints (or an exception to a constraint). The more global the constraints, the better the runtime performance of the tools.

```
# Sample UCF file
# Specify target device and package

CONFIG PART = XC4010e-PQ208-3 ;

# Global constraints

NET CLK1 PERIOD = 40 ;
NET DATA_OUT* OFFSET = OUT 15 AFTER DCLK ;
TIMESPEC TS01 = FROM PADS TO PADS  40 ;
```

```
# Layout constraints

NET SCLINF LOC = P125 ;

# Detailed constraints
# Exception to cover X_DAT and Y_DAT buses

# Ignore timing on reset net

NET RESET_N TIG ;

# Slow exception for data leaving INA FFs

TIMESPEC TS02 = FROM FFS(INA*) TO FFS  80 ;

# Faster timing required for data leaving RAM

TIMESPEC TS03 = FROM RAMS TO FFS 20 ;

# Form special timegroups related to RAMs
INST $1I64 TNM = SPDRAM ;
NET RAMBUS0 TPTHRU = RAMVIA ;
NET RAMBUS1 TPTHRU = RAMVIA ;

# Specify timing for this special timing path

TIMESPEC TS04 = FROM SPDRAM THRU RAMVIA TO FFS 45
  ;
```

Standard Block Delay Symbols

The Table 11-1 lists the block delay symbols, each with their corresponding description. There is a one-to-many correspondence between these symbol names and the *Programmable Logic Data Book* symbol names. For those symbols listed with a disabled default, no timing analysis is performed on paths that have a segment composed of symbol path. For example, paths which have a set/reset to output path will not be analyzed. Any of the block delays (Symbol) listed in the table may be explicitly enabled or disabled using the PCF file.

The following example shows the PCF syntax that enables the path tracing for all paths that contain RAM data to out paths. This PCF directive is placed in the user section of the PCF.

```
SCHEMATIC END;

// This is a PCF comment line
// Enable RAM data to out path tracing

  ENABLE = ram_d_o;
```

Table 11-1 Timing Symbols and Their Default Values

Symbol	Default	Description
reg_sr_q	Disabled	Set / reset to output propagation delay
lat_d_q	Disabled	Data to output transparent latch delay
ram_d_o	Disabled	RAM data to output propagation delay
ram_we_o	Enabled	RAM write enable to output propagation delay
tbuf_t_o	Enabled	TBUF tristate to output propagation delay
tbuf_i_o	Enabled	TBUF input to output propagation delay
io_pad_I	Enabled	IO pad to input propagation delay
io_t_pad	Enabled	IO tristate to pad propagation delay
req_sr_clk	Disabled	Set / Reset to clock setup and hold checks
io_o_I	Enabled	IO output to input propagation delay (Disabled for tristated IOBs.)
io_o_pad	Enabled	IO output to pad propagation delay

Table of Supported Constraints

The following table summarizes all supported constraints; it also shows whether the constraint must be entered at the schematic level or whether it can be specified in one or more of the valid constraint file types (NCF, UCF, or PCF). For further explanation and examples of each of the constraints, see the online *Libraries Guide* (Chapter 12, "Attributes, Constraints, and Carry Logic").

Certain constraints can only be defined at the design level, whereas other constraints can be defined in the various configuration files. The following table lists the constraints and their applicability to the design, and the NCF, UCF, and PCF files.

The CE column indicates which constraints can be entered using the Xilinx Constraints Editor, a GUI tool in the Xilinx Development System. The Constraints Editor passes these constraints to the implementation tools through a UCF file.

An X indicates that the constraint applies to the item for that column.

Note Although the ECS PlaceAndRoute and PlaceAndRoute2 attributes can be entered on the ECS schematic, Xilinx recommends that you enter these attributes in the UCF file instead.

Table 11-2 Constraint Applicability Table

Attribute/ Constraint	ECS Attribute[1]	Design	NCF	UCF	CE	PCF
BLKNM	PlaceAndRoute2	X	X	X		
BUFG	PlaceAndRoute	X	X	X		
CLKDV_DIVIDE	PlaceAndRoute2	X	X	X		
COLLAPSE	PlaceAndRoute	X	X	X		
COMPGRP						X
DECODE	PlaceAndRoute2	X	X	X		
DOUBLE	PlaceAndRoute2	X				
DRIVE	PlaceAndRoute	X	X	X	X	
DROP_SPEC			X	X		X [2]

Table 11-2 Constraint Applicability Table

Attribute/ Constraint	ECS Attribute[1]	Design	NCF	UCF	CE	PCF
DUTY_CYCLE_ CORRECTION	PlaceAndRoute2	X	X	X		
FAST	PlaceAndRoute	X	X	X	X	
FILE		X				
FREQUENCY						X
HBLKNM	PlaceAndRoute2	X	X	X		
HU_SET	PlaceAndRoute2	X	X	X		
INIT	INIT	X	X	X[3]		
INIT_0x	INIT_xx	X	X	X		
INITP_xx	INITP_xx	X	X	X		
IOB	PlaceAndRoute2	X	X	X		
IOSTANDARD	PlaceAndRoute	X	X	X	X	
KEEP	PlaceAndRoute	X	X	X		
KEEPER	PlaceAndRoute	X	X	X	X	
LOC	PlaceAndRoute2	X	X	X	X	X[2]
LOCATE						X
LOCK						X
MAP	PlaceAndRoute2	X	X	X		
MAXDELAY	PlaceAndRoute	X	X	X		X[2]
MAXSKEW	PlaceAndRoute	X	X	X		X[2]
MEDDELAY	PlaceAndRoute	X	X	X		
NODELAY	PlaceAndRoute	X	X	X		
NOREDUCE	PlaceAndRoute	X	X	X		
OFFSET	PlaceAndRoute		X	X	X	X[2]
ONESHOT		X				
OPT_EFFORT	PlaceAndRoute2	X	X	X		
OPTIMIZE	PlaceAndRoute2	X	X	X		
PATH						X

Table 11-2 Constraint Applicability Table

Attribute/ Constraint	ECS Attribute[1]	Design	NCF	UCF	CE	PCF
PART		X	X	X		
PENALIZE TILDE						X
PERIOD	PlaceAndRoute	X	X	X	X	X [2]
PIN						X
PRIORITIZE						X
PROHIBIT		X	X	X	X	X [2]
PULLDOWN	PlaceAndRoute	X	X	X	X	
PULLUP	PlaceAndRoute	X	X	X	X	
PWR_MODE	PlaceAndRoute	X	X	X		
REG	PlaceAndRoute2	X	X	X		
RLOC	PlaceAndRoute2	X	X	X		
RLOC_ORIGIN	PlaceAndRoute2	X	X	X		X
RLOC_RANGE	PlaceAndRoute2	X	X	X		X
S(ave) - Net Flag attribute	PlaceAndRoute	X	X	X		
SITEGRP						X
SLOW	PlaceAndRoute	X	X	X	X	
STARTUP_WAIT	PlaceAndRoute2	X	X	X		
TEMPERATURE		X	X	X	X	X
TIG	PlaceAndRoute	X	X	X	X	X [2]
Time group attributes		X	X	X	X	X
TNM	PlaceAndRoute	X	X	X	X	
TNM_NET	PlaceAndRoute	X	X	X	X	
TPSYNC	PlaceAndRoute	X	X	X		
TPTHRU	PlaceAndRoute	X	X	X	X	
TSidentifier		X	X	X	X	X [2]
U_SET	PlaceAndRoute2	X	X	X		
USE_RLOC	PlaceAndRoute2	X	X	X		

Table 11-2 Constraint Applicability Table

Attribute/ Constraint	ECS Attribute[1]	Design	NCF	UCF	CE	PCF
VOLTAGE		X	X	X	X	X
WIREAND	PlaceAndRoute2	X	X	X		
XBLKNM	PlaceAndRoute2	X	X	X		

NOTES:

[1]ECS attributes can be applied to symbols or nets as follows: The PlaceAndRoute2 attribute is set on a symbol via the Symbol Editor. The PlaceAndRoute attribute is set on nets via the Schematic Editor for projects that use XST synthesis tools. (Although the ECS PlaceAndRoute and PlaceAndRoute2 attributes can be entered on the ECS schematic, Xilinx recommends that you enter these attributes in the UCF file instead.)

[2]Use cautiously — although the constraint is available, there are differences between the UCF / NCF and PCF syntax.

[3]INIT is allowed in the UCF for CPLDs only.

Basic UCF Syntax Examples

The following sections summarize the functions of timespecs.

PERIOD Timespec

The PERIOD spec covers all timing paths that start or end at a register, latch, or synchronous RAM which are clocked by the reference net (excluding pad destinations). Also covered is the setup requirement of the synchronous element relative to other elements (for example, flip flops, pads, and so forth).

Note The default unit for time is nanoseconds.

```
NET clk20MHz PERIOD = 50 ;
NET clk50mhz TNM = clk50mhz ;
TIMESPEC TS01 = PERIOD : clk50mhz : 20 ;
```

FROM:TO Timespecs

FROM:TO style timespecs can be used to constrain paths between time groups.

Note Keywords: RAMS, FFS, PADS, and LATCHES are predefined time groups used to specify all elements of each type in a design.

```
TIMESPEC TS02 = FROM : PADS : TO : FFS : 36 ;
TIMESPEC TS03 = FROM : FFS : TO : PADS : 36 ns ;
TIMESPEC TS04 = FROM : PADS : TO : PADS : 66 ;
TIMESPEC TS05 = FROM : PADS : TO : RAMS : 36 ;
TIMESPEC TS06 = FROM : RAMS : TO : PADS : 35.5 ;
```

OFFSET Timespec

To automatically include clock buffer / routing delay in your "PADS:TO: *synchronous element* or *synchronous element* :TO:PADS timing specifications, use OFFSET constraints instead of FROM:TO constraints.

- For an input where the maximum clock-to-out (Tco) of the driving device is 10 ns.

  ```
  NET in_net_name OFFSET=IN:10:AFTER:clk_net ;
  ```

- For an output where the minimum setup time (Tsu) of the device being driven is 5 ns.

  ```
  NET out_net_name OFFSET=OUT:5:BEFORE:clk_net ;
  ```

Timing Ignore

If you can ignore timing of paths, use Timing Ignore (TIG).

Note The "*" character is a wild-card which can be used for bus names. A "?" character can be used to wild-card one character.

- Ignore timing of net *reset_n*:

  NET : reset_n : TIG ;

- Ignore *data_reg(7:0)* net in instance *mux_mem*:

  NET : mux_mem / data_reg* : TIG ;

- Ignore *data_reg(7:0)* net in instance *mux_mem* as related to a TIMESPEC named TS01 only:

  NET : mux_mem / data_reg* : TIG = TS01 ;

- Ignore *data1_sig* and *data2_sig* nets:

Path Exceptions

If your design has outputs that can be slower than others, you can create specific timespecs similar to this example for output nets named *out_data(7:0)* and *irq_n*.

```
TIMEGRP slow_outs = PADS(out_data* : irq_n) ;
TIMEGRP fast_outs = PADS : EXCEPT : slow_outs ;
TIMESPEC TS08 = FROM : FFS : TO : fast_outs : 22 ;
TIMESPEC TS09 = FROM : FFS : TO : slow_outs : 75 ;
```

If you have multi-cycle FF to FF paths, you can create a time group using either the TIMEGRP or TNM statements.


Warning

Many VHDL / verilog synthesizers do not predictably name flip flop Q output nets. Most synthesizers do assign predictable instance names to flip flops, however.

- TIMEGRP example.

  ```
  TIMEGRP slowffs = FFS(inst_path/
     ff_q_output_net1* : inst_path/
     ff_q_output_net2*);
  ```

- TNM attached to instance example.

  ```
  INST inst_path/ff_instance_name1_reg* TNM =
     slowffs ;

  INST inst_path/ff_instance_name2_reg* TNM =
     slowffs ;
  ```

- If a FF clock-enable is used on all flip flops of a multi-cycle path, you can attach TNM to the clock enable net.

  Note TNM attached to a net “forward traces” to any FF, LATCH, RAM, or PAD attached to the net.

```
NET ff_clock_enable_net TNM = slowffs ;
```

- Example of using “slowffs” timegroup, in a FROM:TO timespec, with either of the three timegroup methods previously shown.

```
TIMESPEC TS10 = FROM : slowffs : TO : FFS : 100 ;
```

Miscellaneous Examples

- Assign an IO pin number or place a basic element (BEL) in a specific CLB. BEL = FF, LUT, RAM, etc...

```
INST io_buf_instance_name LOC = P110 ;

NET io_net_name LOC = P111 ;

INST instance_path/BEL_inst_name LOC =
   CLB_R17C36 ;
```

- Prohibit IO pin C26 or CLB_R5C3 from being used.

```
CONFIG PROHIBIT = C26 ;

CONFIG PROHIBIT = CLB_R5C3 ;
```

- Assign an OBUF to be FAST or SLOW.

```
INST obuf_instance_name FAST ;

INST obuf_instance_name SLOW ;
```

- Constrain the skew or delay associate with a net.

```
NET any_net_name MAXSKEW = 7 ;

NET any_net_name MAXDELAY = 20 ns;
```

- Declare an IOB input FF delay (default = MAXDELAY).

  Note MEDDELAY / NODELAY can be attached to a CLB FF that is pushed into an IOB by the “map -pr i” option.

```
INST input_ff_instance_name MEDDELAY ;

INST input_ff_instance_name NODELAY ;
```

- Also, constraint priority in your .ucf file is as follows:

  Highest

  1. Timing Ignore (TIG)
  2. FROM : THRU : TO specs
  3. FROM : TO specs lowest
  4. OFFSET
  5. PERIOD specs

  Lowest

See the on-line documentation set for additional timespec features or additional information.

Chapter 12

Simulation

Simulation verifies the operation of your design before you implement it as hardware. It allows you to observe the circuit's behavior at its inputs and outputs as well as the behavior of internal nodes. You can use testbenches (VHDL) or test fixtures (Verilog) to specify circuit input stimuli and output responses and then test to those specifications at various points in the design flow. For simulation functions, Foundation Series ISE includes integrated support for ModelSim simulators from Model Technology Incorporated (MTI) and for HDL Bencher, a testbench / test fixture generating tool from Visual Software Solutions (VSS). This chapter describes the integration of the ModelSim simulators and HDL Bencher with Foundation Series ISE. It contains the following sections:

- "ModelSim"
- "Acquiring ModelSim Tools"
- "Launching ModelSim"
- "Functional Simulation"
- "Timing Simulation (Post-Route)"
- "Creating a Testbench / Test Fixture"

ModelSim

The ModelSim simulation tools from Model Technology Incorporated (MTI) provide a complete HDL simulation and debugging environment for your Foundation Series ISE project. ModelSim simulators support simulation of VHDL and Verilog HDL designs. Some editions (SE) support mixed-HDL designs (Verilog and VHDL). Source code debugging, functional simulation, and back-annotated timing simulation are supported in all editions.

ModelSim provides 100 percent VHDL and Verilog language coverage, a source code viewer / editor, waveform viewer, design structure browser, list window and many other features designed to enhance productivity.

ModelSim VHDL supports both the IEEE 1076-1987 and 1076-1993 VHDL, 1164-1993 Standard Multivalue Logic System for VHDL Interoperability, and the 1076.2-1996 Standard VHDL Mathematical Packages standards.

ModelSim Verilog is based on the IEEE Std 1364-1995 Standard Hardware Description Language Based on the Verilog Hardware Description Language. The Open Verilog International Verilog LRM version 2.0 is also applicable to a large extent. Both PLI (Programming Language Interface) and VCD (Value Change Dump) are supported for ModelSim PE and EE users.

In addition, all products support SDF 1.0 through 3.0, VITAL 2.2b, and VITAL'95 - IEEE 1076.4-1995.

Acquiring ModelSim Tools

The Foundation Series ISE 3.1i installation does not automatically install the ModelSim tools. You can acquire and / or access these tools as described in the following sections.

ModelSim XE Starter

The Starter version of ModelSim XE (ModelSim XE Starter) is included free on the MTI CD you received with your Foundation Series ISE 3.1i software package. You can license it through the Xilinx support website by using the link on the installation screen.

ModelSim XE Starter supports both VHDL and Verilog but can only simulate one language at a time. It features pre-compiled Xilinx libraries and can handle up to 500 lines of code, including the testbench or test fixture. If you are new to HDL design, ModelSim XE Starter enables you to experiment with and create small VHDL designs and testbenches or Verilog designs and test fixtures.

ModelSim XE

The Xilinx Edition of ModelSim (ModelSim XE) is a limited version of MTI's ModelSim tools. ModelSim XE is available in a VHDL or Verilog version. Mixed HDL designs are not supported. It features pre-compiled Xilinx libraries and can handle about 100,000 system gates depending on the size of the design's testbench or test fixture. This is usually sufficient for designs targeting Xilinx XC9500 CPLDs and Spartan FPGAs as well as the medium-density XC4000 and Virtex FPGAs.

An evaluation version of ModelSim XE is included on the MTI CD you received with your Foundation Series ISE 3.1i software package. It does require licensing. Use the following procedure to install the Xilinx Edition of ModelSim:

1. Insert your ModelSim Xilinx Edition CD into your PC's CD-ROM drive.
2. In your Windows Start menu, select, **`Start`** → **`Programs`** → **`Xilinx Foundation Series ISE 3.1i`** → **`Partner Products`** → **`Install ModelSim Xilinx Edition (CD Required)`**.
3. Follow the directions in the installation screens. You are asked whether you want to install the Evaluation or Full version of XE.

   The Evaluation version installer installs the software and allows you to apply for a one-time, 30-day evaluation license on the Web. The Full version installer performs a similar installation, and allows you to apply for a permanent license through the Web, but you must have already purchased the software from Xilinx for the license application to succeed.
4. Because ModelSim XE supports either VHDL or Verilog simulation, you must choose one language to install.

You can contact Xilinx Customer Service for purchase and licensing of the ModelSim XE product. Call the Xilinx hotline for ModelSim XE support.

Additional ModelSim Editions

The following products are also supported in Foundation Series ISE:

- ModelSim PE (Personal Edition)
- ModelSim SE (Special Edition)

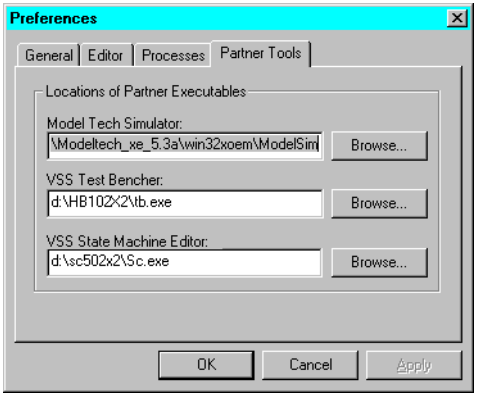
Evaluation version of ModelSim PE and SE are included on the MTI CD you received with your Foundation Series ISE 3.1i software package. You can license these products through MTI for free evaluation for up to 30 days.

Contact MTI for information on the ModelSim PE and SE configurations. All product sales and licensing for these products are handled directly by MTI and its authorized sales affiliates (e-mail sales@model.com). Customer support is also provided directly by MTI (e-mail support@model.com or call the main number at (503) 641-1340).

Previously Installed ModelSim Tools

If you were using ModelSim tools before you installed Foundation Series ISE, you can direct the Project Navigator to that ModelSim installation. Use the following procedure to designate any pre-existing ModelSim tools you want to use with Foundation Series ISE:

1. From the Project Navigator Main Menu, select **`Edit`** → **`Preferences`**.
2. In the Preferences window, select the **`Partner Tools`** tab.



3. In the **Model Tech Simulator** field on the Partner Tools tab, enter the path to the program executable for an existing installation of ModelSim. Or, click **Browse** to select the path.

Sources for Learning to use ModelSim

The ModelSim online help (**Help** in the ModelSim main window) contains the following resources to help you learn how to use the ModelSim simulators:

- User's Manual
- Command Reference
- Quick Start Menu

Note The Xilinx support website also contains useful information on using ModelSim with Xilinx products. The URL for this website is `http://www.support.xilinx.com/support/techsup/journals/mti`.

Launching ModelSim

After you install one of the ModelSim tools as described in the "Acquiring ModelSim Tools" section, you can launch it from within the Project Navigator.

Foundation Series ISE supports two simulation points in its design process: functional simulation and timing simulation.

- Functional Simulation (RTL Level)

  The first simulation point is functional simulation at the Register Transfer Level (RTL). Functional simulation at this point is used to verify that the logic of your design functions as intended. RTL functional simulation is performed after design entry but prior to synthesis.

  Use one of the following methods to launch ModelSim from the Project Navigator for functional simulation:

  - Highlight the desired source file in the Sources window and then select **Launch ModelSim Simulator** in the Processes window.

- Highlight a testbench / test fixture file in the Sources window and then double-click on **Simulate Functional VHDL/ Verilog Module** in the Processes window.

Refer to the "Functional Simulation" section for more information on functional simulation with ModelSim.

- Post -Route Timing Simulation

  The second simulation point is timing simulation. Timing simulation includes detailed timing information for the targeted device. Timing simulation is performed after synthesis and place and route. It always requires a a testbench or test fixture.

  To launch ModelSim from the Project Navigator for timing simulation, highlight a testbench / test fixture file in the Sources window and then double-click **Simulate Post-Route VHDL/ Verilog Module** in the Processes window.

  Refer to the "Timing Simulation (Post-Route)" section for more information on timing simulation with ModelSim.

ModelSim Integration Overview

The Project Navigator creates a ModelSim Do file (a macro file) that performs the following functions *automatically* when you invoke ModelSim simulation for your project:

- Launch ModelSim with default or user-specified controls.

  You can set simulation Process Properties for launching ModelSim in the Project Navigator, if desired. These include properties for simulation runtime, ModelSim windows to display, whether to use a user-supplied Do file, etc. Refer to the "Functional Simulation Process Properties" section and "Timing Simulation Process Properties" section for simulation Process Properties information.

- Create and map a working library (**work**) that ModelSim can use to run a simulation.
- Map the required simulation libraries (depending on the type of simulation being run and device being used).
- Generate an HDL model for all non-HDL source files. For schematic sources, this means netlisting them. ABEL sources are translated into structural HDL.

- Compile all of the HDL source files comprising the design selected for simulation into the working library
- Run a simulation for 1000 time units (or as specified in the simulation Process Properties) and displays the results in ModelSim's Wave and List windows.

ModelSim opens with the specified design loaded and ready for you to take control of the simulation to debug and verify your design. You can edit any of the HDL source files using ModelSim's source editor. You can exit ModelSim whenever you want.

If you do not want Foundation Series ISe to initialize ModelSim as described in this section, you can "turn off" the automatic initialization by unchecking the **Use Automatic Do File** box in the synthesis Process Properties dialog box for Simulate Functional VHDL Model or Simulate Post-Route VHDL Model. Refer to the "Functional Simulation Process Properties" section for information on setting synthesis properties.

Simulator Initialization and VHDL Package Sources

VHDL Packages are treated differently than other VHDL files. Packages (ieee.std_logic_1164, for example) are often used by many other VHDL design units. Each time a package is recompiled, all other design units that use it must also be recompiled. Needless to say, it is not desirable for packages to be constantly recompiled. Therefore, the Do files generated by Foundation Series ISE do not attempt to compile VHDL Package sources in the installed simulator when functional or post-route simulation is performed.

Use the following procedure to compile a VHDL Package source for simulation:

1. Select a VHDL package in the Sources window.
2. Select **Compile For Simulation** in the Processes window.

Xilinx Simulation Libraries

Xilinx provides the following libraries for ModelSim simulation flows.

- unisim (UNISIM, UNISIM5K for VHDL; UNISIM_VER, UNI3000, UNI5200, UNI9000 for Verilog)
- simprim (SIMPRIM for VHDL and SIMPRIM_VER for Verilog)
- logiblox
- corelib (xilinxcorelib for VHDL; xilinxcorelib_ver for Verilog)
- Virtex_Macro
- Virtex2_macro
- Spartan2_macro
- XC9000_macro
- XC4000_macro

For ModelSim XE, these libraries are pre-compiled and ready for use. For other ModelSim editions, these libraries need to be compiled. Please refer to Solution 2561 on the Xilinx support website `http://www.support.xilinx.com` for information on compiling these Xilinx libraries for use with Foundation Series ISE.

Functional Simulation

Functional simulation is the first simulation point in the Foundation Series ISE design flow. It is performed at the Register Transfer Level (RTL). The main purpose of RTL simulation is to verify the HDL syntax and determine if the behavior of the HDL code is what is expected.

Functional simulation is run after design entry and prior to synthesis. Simulation is done using the HDL code that describes the function of the design or a testbench / test fixture. The design at this point is device and implementation independent. Therefore, no timing information is available.

Most design development is usually done through iterative RTL simulation until the final functionality is achieved. The simulation can be either started from a testbench / test fixture or from an individual HDL source module.

For testbench / test fixture simulation, the testbench / test fixture may be automatically generated. You then supply the test stimuli.

For simulations without a testbench / test fixture, you may drive the simulation using ModelSim's command console or write a simulator control file (do file).

Functional Simulation with a Testbench/Test Fixture

You can create a testbench or test fixture to use for stimuli in your design as follows:

1. Create a testbench / test fixture as described in "Creating a Testbench / Test Fixture" section
2. Add the testbench / test fixture file to the project as described in the "Adding the Testbench / Test Fixture File to the Project" section.
3. Click on the testbench / test fixture file in the "Sources in Project" window.
4. In the "Processes for Current Source" window, double-click on **`Simulate Functional VHDL Module`** under the ModelSim Simulator.
5. The ModelSim simulator opens with the files compiled and the design loaded.

By default, the ModelSim simulator opens with its command window, Display Signals window, Display Structures window, and Display Wave window. You can control which ModelSim windows open initially by setting simulation process properties in the Project Navigator. Refer to the "Functional Simulation Process Properties" section for information on setting simulation process properties.

Interactive Functional Simulation

For simulations without a testbench / test fixture, you can drive the simulation using ModelSim's command console or a user generated simulator control (.do) file.

Simulating with ModelSim's Command Console

Use the following procedure to simulate the design interactively:

1. Click on a source file (not a testbench file) in the Sources window.
2. Select **`Launch MTI Simulator`** under Design Entry Utilities in the Processes window.
3. The ModelSim simulator opens with source files compiled and loaded. The libraries must be compiled separately.

   By default, the ModelSim simulator opens with its command window, Display Signals window, Display Structures window, and Display Wave window as shown in the following figure. You can control which ModelSim windows open initially by setting simulation process properties in the Project Navigator. Refer to the "Functional Simulation Process Properties" section for information on setting simulation process properties.

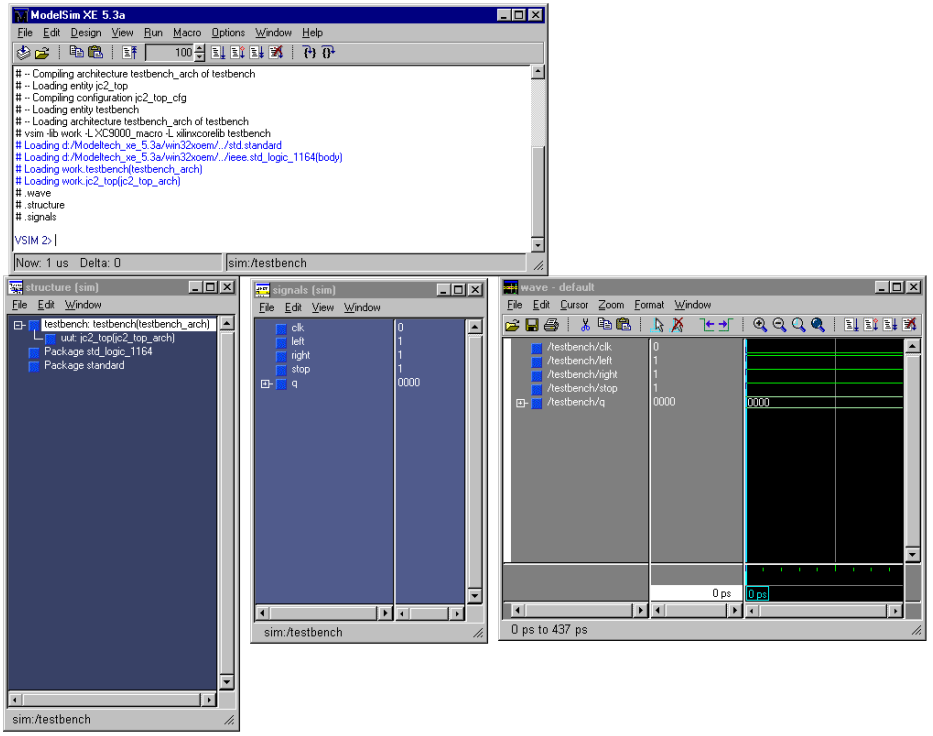


Figure 12-1 ModelSIM Default Windows

Simulating with Do Files

Two types of do files are available. You can enter simulation commands in the *testbench_file_name*.zdo file and compilation commands in the *testbench_file_name*.udo file. These files are automatically created the first time the test bench is simulated.

Functional Simulation Process Properties

You can specify process properties for functional simulation. Once selected, the process properties apply whenever ModelSim is launched for functional simulation by the Project Navigator.

HDL Source Module

To set properties for functional simulation of an HDL source module, use the following procedure:

1. Click on an HDL source module in the Sources Window.
2. Right-click on **`Launch MTI Simulator`** under Design Entry Utilities in the Processes window.
3. Select **`Properties`** from the list box that appears.
4. For VHDL sources, a Process Properties dialog box with the VHDL Simulation Options tab appears. For Verilog, the Verilog Simulation Options tab appears. Both tabs contain the same options.

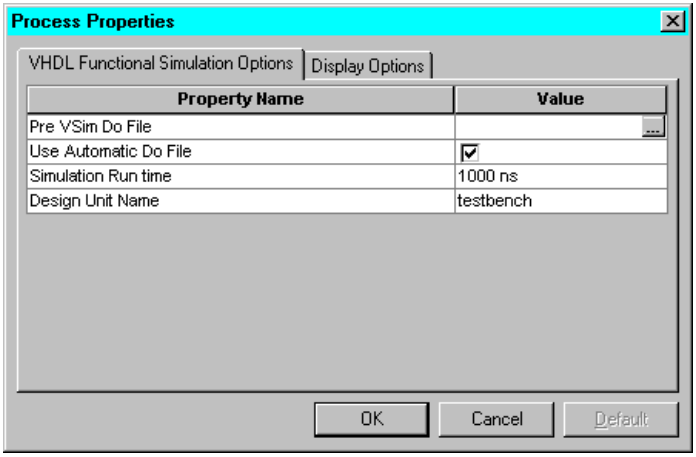
Testbench/Test Fixture

To set properties for functional simulation of an HDL testbench / test fixture, use the following procedure:

1. Click on an HDL testbench / test fixture in the Sources Window.
2. Right-click on **`Simulate Functional VHDL Model`** under ModelSim Simulator in the Processes window.
3. Select **`Properties`** from the list box that appears.
4. For VHDL sources, a Process Properties dialog box with the VHDL Simulation Options tab appears. For Verilog, the Verilog Simulation Options tab appears. Both tabs contain the same options.

VHDL/Verilog Simulation Options Tab

The properties you can set on the VHDL or Verilog Simulation Options tab (shown in the following figure) are listed and described in the following sections.



Pre VSim do File

This option allows you to enter the name of a do file to be run before the VSIM command in ModelSim. The do file specified here can only contain VCOM commands for VHDL designs or VLOG commands for Verilog designs. Refer to Table 12-1 and Table 12-2 for the VCOM and VLOG command that can be included in a Pre VSim do File.

Table 12-1 Valid VCOM Commands for Pre VSim do File (VHDL)

Key Argument	Description
[-help]	Displays **vcom** syntax help
[-version]	Returns **vcom** version
[-93][-87]	Selects VHDL 1993 or 1987
[-explicit]	Resolves ambiguous overloads
[-f <*filename*>]	Passes in arguments from file <*filename*>
[-nocheck]	Disables run time range checks

Table 12-1 Valid VCOM Commands for Pre VSim do File (VHDL)

Key Argument	Description
[-nowarn <#>]	Disables individual warning message (<#>)
[-O0]	Disables optimization
[-quiet]	Disables loading messages
[-refresh]	Regenerates library image
[-work <*libname*>]	Specifies **work** library
<*filename(s)*>	Specifies VHDL file(s) (<*filename(s)*>) to be compiled
Examples: vcom MyDesign.vhd vcom -93 -work /lib/mylib util.vhd vcom -refresh	

Table 12-2 Valid VLOG Commands for Pre VSim do File (Verilog)

Key Argument	Description
[-help]	Displays **vlog** syntax help
[-version]	Returns **vlog** version
[-f <*filename*>]	Passes in arguments from file <*filename*>
[-hazards]	Enables runtime hazard checking
[-nodebug]	Hides internal variables and structure
[-quiet]	Disables loading messages
[-R <*simargs*>]	Invokes VSIM after compile
[-refresh]	Regenerates lib to current version
[-work <*libname*>]	Specifies **work** library
[-v <*library_file*>]	Specifies Verilog source library

Table 12-2 Valid VLOG Commands for Pre VSim do File (Verilog)

Key Argument	Description
<*filename(s)*>	Specifies Verilog file(s) (<*filename(s)*>) to be compiled
Examples: vlog top.v vlog -work mylib -refresh	

Use Automatic do File

When enabled (a check mark in the Value box), the Use Automatic Do File property automatically generates a macro file that compiles all VHDL files in your project, starts the simulation process, and initializes the waveform display.

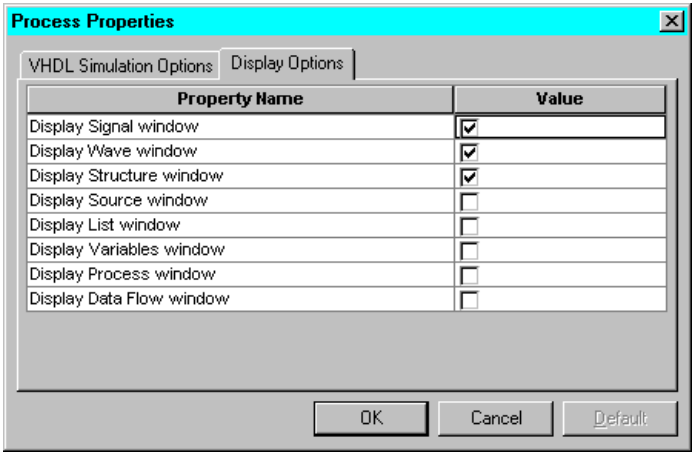
If desired, you may add additional initialization commands in the file called *testbench_file_name*.udo. The file is automatically created the first time the test bench is simulated.

Simulation Run Time

The Simulation Run Time option appears for testbenches / test fixtures only. Specify how long the simulation should run. The default is 1000 ns. At the end of the runtime, the wave window displays the results of the simulation.

Display Options Tab

You can control which ModelSim windows appear when the ModelSim simulator opens using the selections on the Options tab (shown in the following figure).



Display Signals Window

Place a check mark (by clicking in the box) in the Value field to have the ModelSim Signals window display when ModelSim is invoked by the Project Navigator. The default is to display the Signals window, which shows the names and current values of signals.

Display Wave Window

Place a check mark (by clicking in the box) in the Value field to have the ModelSim Wave window display when ModelSim is invoked by the Project Navigator. The default is to display the Wave window, which shows a graphical view of the simulation results.

Display Structure Window

Place a check mark (by clicking in the box) in the Value field to have the ModelSim Structure window display when ModelSim is invoked by the Project Navigator. The default is to display the Structure window, which shows a hierarchical view of the design.

Display Source Window

Place a check mark (by clicking in the box) in the Value field to have the ModelSim Source window display when ModelSim is invoked by the Project Navigator. By default, the Source window, which contains the VHDL or Verilog source code, is not displayed.

Display List Window

Place a check mark (by clicking in the box) in the Value field to have the ModelSim List window display when ModelSim is invoked by the Project Navigator. By default, the List window, which displays simulation results in table form, is not displayed.

Display Variables Window

Place a check mark (by clicking in the box) in the Value field to have the ModelSim Variables window display when ModelSim is invoked by the Project Navigator. By default, the Variables window, which shows generics, constants, and variables for VHDL and shows registers for Verilog, is not displayed.

Display Process Window

Place a check mark (by clicking in the box) in the Value field to have the ModelSim Process window display when ModelSim is invoked by the Project Navigator. By default, the Process window, which shows the run schedule for VHDL and Verilog processes, is not displayed.

Display Data Flow Window

Place a check mark (by clicking in the box) in the Value field to have the ModelSim Data Flow window display when ModelSim is invoked by the Project Navigator. By default, the Data Flow window, which allows design tracing of VHDL signals and design tracing of Verilog nets or registers, is not displayed.

Timing Simulation (Post-Route)

Timing simulation is performed after synthesis and Place and Route, using back-annotation information. The back annotation processes generate a netlist of library components annotated in a Standard Delay Format (SDF) file.

Timing simulation verifies that your design runs at the desired speed for your device under worst-case conditions. It can verify timing relationships and determine the critical paths for the design under worst-case conditions. It can also determine whether the design contains set-up or hold violations.

The procedures for functional and timing simulation are nearly identical. Functional simulation is performed *before* the design is placed and routed and simulates only the functionality of the logic in the design. Timing simulation is performed *after* the design is placed and routed and uses timing information based on the delays in the placed and routed design. Timing simulation describes the circuit behavior far more accurately than functional simulation.

Like functional simulation, you must use input stimulus to run the simulation.

Note For schematic designs, naming the nets during your design entry is very important for both functional and timing simulation. This allows you to find the nets in the simulations more easily than looking for a machine-generated name.

For timing simulation, all logic in the design is converted to simulation primitives (SIMPRIMS). Logic converted to simprims can be simulated by the Xilinx Verilog or VITAL simulation libraries.

Automatic Macro File Generation and Post-Route Simulation

To initialize the simulator for post-route simulation, ISE creates a macro file called *testbench_file_name*.tdo. This file is similar to the .fdo file, except that it compiles the post-route VHDL netlist prior to compiling the testbench. Do not edit the .tdo file.

Disabling Automatic Macro File Generation

If you do not wish to use the automatic macro file generation process and automatic initialization of the simulator, then uncheck the Use Automatic Do File property for Simulate Functional VHDL Model or Simulate Post-Route VHDL Model.

Simulating with a Testbench

Timing simulation is started from a testbench (ideally the same one used for functional simulation). To run a gate-level simulation from a testbench, do the following:

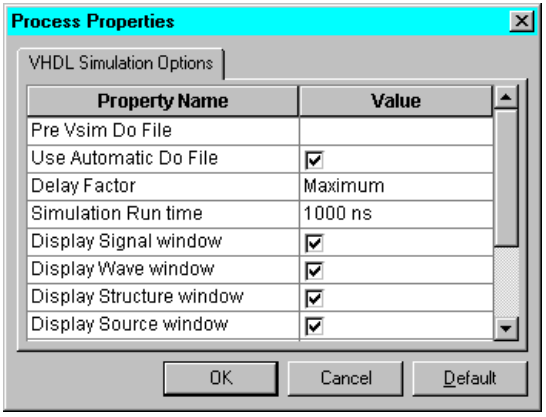
1. If you do not have an existing testbench / test fixture, create one using one of the methods described in "Creating a Testbench / Test Fixture" section. Add the testbench / test fixture file to the project as described in the "Adding the Testbench / Test Fixture File to the Project" section.
2. Click on the testbench / test fixture file in the "Sources in Project" window.
3. Under "ModelSim Simulator" in the "Processes for Current Source" window, select **`Simulate Post Route VHDL Module`** or **`Simulate Post Route Verilog Module`**. (The appropriate selection appears depending on whether a VHDL or Verilog source module is selected.)
4. The ModelSim simulator opens with the source files compiled and the design loaded. The testbench or test fixture is also compiled and loaded.

   By default, the ModelSim simulator opens with its command window, Display Signals window, Display Structures window, and Display Wave window. You can control which ModelSim windows open initially by setting ModelSim process properties. Refer to the "Functional Simulation Process Properties" section for information on setting ModelSim process properties.

Timing Simulation Process Properties

To set properties for Timing simulation of an HDL testbench / test fixture, use the following procedure:

1. Click on an HDL testbench / test fixture in the Sources Window.
2. Right-click on **`Simulate Post-Route VHDL Model`** under ModelSim Simulator in the Processes window.
3. Select Properties from the list box that appears.
4. For VHDL sources, the VHDL Process Properties dialog box appears. For Verilog, the Verilog Simulation Options dialog box appears. Both dialog boxes contain the same options.



Timing Simulation Options

The properties you can set for the Timing simulation process are the same as those listed for functional simulation (see the “Functional Simulation Process Properties” section) plus one additional property: Design Unit Name.

Design Unit Name

The Design Unit Name property allows you to specify the name of the design unit you want to simulate. By default, this field is set to “testbench”.

Creating a Testbench/Test Fixture

A testbench / test fixture is a separate file with VHDL or Verilog code that connects to the inputs and outputs of a design under test. In a testbench / test fixture, you can provide stimuli and response information that can be used for running ModelSim simulation.

There are three ways you can create a testbench (VHDL) or test fixture (Verilog) for simulation:

1. Use the provided testbench generator tool HDL Bencher ™ from Visual Software Solutions (VSS). (Refer to the "Using StateBench" section of the "State Diagrams" chapter for information.)
2. Write it from scratch.
3. Use the Testbench / Test Fixture template generator provided in the ISE Project Navigator.

HDL Bencher

HDL Bencher is a testbench / test fixture creation tool from Visual Software Solutions (VSS).

Acquiring HDL Bencher

The Foundation Series ISE 3.1i installation does not automatically install the HDL Bencher. You can acquire and / or access these tools as described in the following sections.

All sales, support, and licensing of HDL Bencher is done through VSS, Inc. This includes support for the Xilinx Edition. Go to their website at `http://www.testbench.com/order.htm` or contact them at (800) 208-1051.

All Foundation Series ISE users qualify for a $500 discount on the purchase of the HDL Bencher Complete Edition. To receive the discount, you need to provide your Host ID, which is located in HDL Bencher's **`Help`** → **`About`** dialog box. To purchase the HDL Bencher Complete Edition, contact Visual Software Solutions at (954) 370-9030 or e-mail sales@testbench.com.

Xilinx Edition

The Xilinx Edition of HDL Bencher is provided free with your Foundation Series ISE software. You must install it from the VSS CD to make it available from the Project Navigator.

The Xilinx Edition of HDL Bencher can handle 12 signal assignments, 6 port signals, and 8-bit maximum bus width. If you register with VSS when you install the Xilinx Edition, the limits for the Xilinx Edition of HDL Bencher are increased to 21 signal assignments, 7 port signals, and 8-bit maximum bus width.

Signal assignments are counted any time a signal explicitly gets assigned after the initial (clock cycle 0) assignment. Clock transitions are not counted. A bus changing state counts as one signal assignment. *Port signals* are each port declared in the top level entity of the design. A std_logic_vector port type counts as one port signal.

Use the following procedure to install the Xilinx Edition of HDL Bencher:

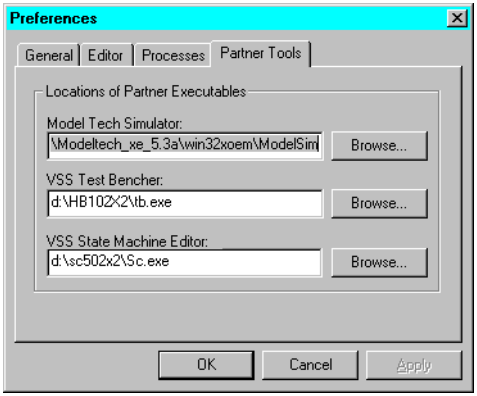
1. Insert your ALLSTAR VSS CD into your PC's CD-ROM drive.
2. In your Windows Start menu, select, **`Start`** → **`Programs`** → **`Xilinx Foundation Series ISE 3.1i`** → **`Partner Products`** → **`Install HDL Bencher Xilinx Edition (CD Required)`**.
3. Follow the directions in the installation screens.

Pre-Existing or Upgraded HDL Bencher Tool

You do not need to install the Xilinx limited edition of HDL Bencher to add automated testbench / test fixture creation to Foundation Series ISE. If you currently have a fully functional version of HDL Bencher or if you obtain it from VSS later, you can instruct Foundation Series ISE to use it with your project.

Use the following procedure to identify the HDL Bencher installation you want to use with Foundation Series ISE:

1. From the Project Navigator Menu bar, select **`Edit`** → **`Preferences`**.
2. In the Preferences window, select the Partner Tools tab.



3. In the **VSS Test Bencher** field on the Partner Tools tab, enter the path to the program executable for an existing installation of StateCAD / StateBench. Or, click **Browse** to select the path.

Complete Edition

You can obtain the unlimited, fully-functional version of HDL Bencher from VSS. Contact them directly for information concerning purchasing and licensing.

All Foundation Series ISE users qualify for a $500 discount on the purchase of the HDL Bencher Complete Edition. To receive the discount, you need to provide your Host ID, which is located in HDL Bencher's **Help** → **About** dialog box. To purchase the HDL Bencher Complete Edition, contact Visual Software Solutions at (954) 370-9030 or e-mail sales@testbench.com.

Launching HDL Bencher

Use the following procedure to create a testbench / test fixture with HDL Bencher:

1. Click on a Verilog, VHDL, or schematic source module in the Sources window.
2. Double-click on **Launch HDL Bencher Tool** under Design Entry Utilities in the Processes window.
3. HDL Bencher opens with the selected source file already loaded as shown in the following figure. Just update the waveform and export it to create a complete, simulatable testbench.

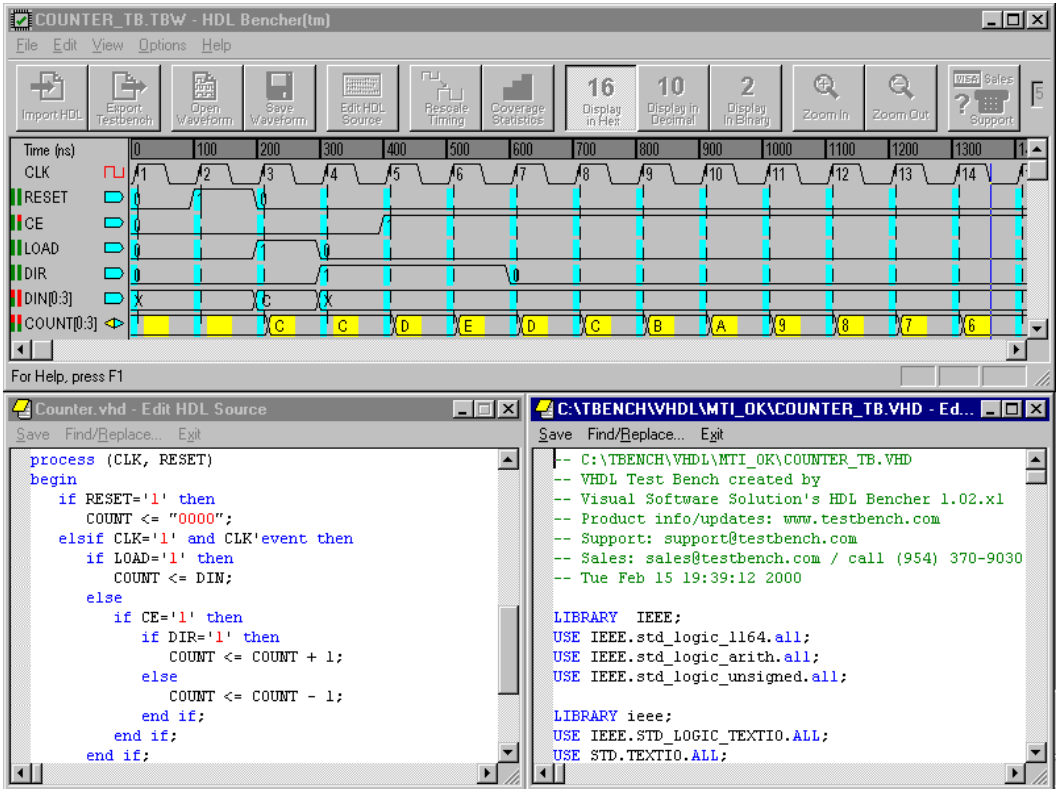


Figure 12-2 HDL Bencher Waveform

Refer to the content-sensitive help and basic tutorial available from the HDL Bencher Help menu to get you started using the HDL Bencher tool.

4. Upon completion of the testbench / test fixture, HDL Bencher writes the testbench / test fixture file into the ISE project directory.

   HDL Bencher also writes a waveform stimulus / response file (.tbw) into the project directory.

5. You must add the testbench / test fixture file to your project as a testbench / test fixture source (see the“Adding the Testbench / Test Fixture File to the Project” section for instructions). You can add the waveform file as a user document.

Testbench / test fixture files written by HDL Bencher work with the ModelSim simulator as described in the "Functional Simulation" section section.

After the testbench / test fixture has been added to the project, it will appear in the Sources window immediately below its associated source file. You can double-click on the testbench / test fixture file to re-open HDL Bencher.

Testbench/Test Fixture Template Generator

Foundation Series ISE includes an automatic testbench generator for VHDL modules and an automatic test fixture generator for Verilog modules. Use the following procedure to generate a testbench or test fixture:

1. Click on a VHDL, Verilog, or schematic source module in the Sources window.
2. For a VHDL testbench, double click on **`View VHDL Testbench Template`** under Design Entry Utilities in the Processes window.

   For a Verilog test fixture, double click on **`View Verilog Test Fixture Declaration`** under Design Entry Utilities in the Processes window.
3. A testbench template or test fixture declaration immediately appears in the Report Viewer for the selected source module. Because the Report Viewer window is read-only, you must do the following to use the template / declaration in your project:

   a) Select **`File`** → **`Save As`** from the Report Viewer menu.

   b) Save the file with any name you choose with the extension .vhd for a VHDL testbench or .v for a Verilog test fixture.

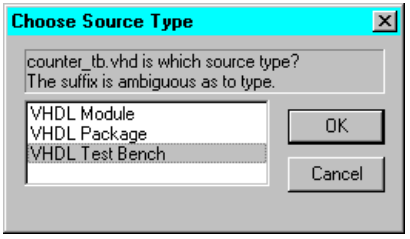
   c) Add the necessary code to complete the testbench / test fixture.

   d) Add the testbench or test fixture to your project (see the "Adding the Testbench / Test Fixture File to the Project" section for instructions).

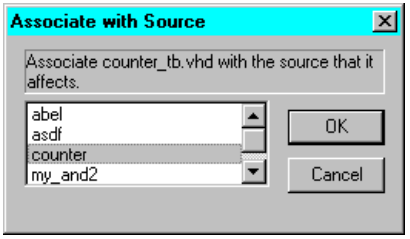
Adding the Testbench/Test Fixture File to the Project

To use a testbench or test fixture for simulation, you must add it to the project after you create it. Use the following procedure to add a testbench / test fixture file to your project.

1. Select **`Project`** → **`Add Source`** from the Project Navigator menu bar.
2. Browse to the testbench / test fixture file to select the testbench / test fixture to be added. Then click **`Open`**.
3. Select **`VHDL Test Bench`** or **`Verilog Test Fixture`**, as appropriate, from the Choose Source Type dialog box. Then click **`OK`**.



4. Select the source module from the list in the Associate with Source dialog box to associate the testbench / test fixture with that source. Then click **`OK`**.



5. The testbench / test fixture is added to the Sources in Project window under the associated source file.

Conventions

Whichever method you use to create a testbench or test fixture for simulation, there are certain conventions that should be followed to ensure that it works properly with Foundation Series ISE.

Testbench Naming Conventions

ISE relies on the following testbench naming conventions:

- The entity name of a testbench should be **`testbench`**.
- The architecture name of a testbench should be **`behavior`**.
- The instance name of the component instance for the unit under test (the design being simulated) should be **`uut`**.

All testbench templates generated by ISE follow these naming conventions. You are free to ignore these conventions if you want. However, if you do, ISE will not automatically initialize the simulator. (See the next section for details on this feature). Also, post-route simulation may be restricted.

Port Type Requirements

With ISE, you should use the IEEE 1164 types std_logic and std_logic_vector for the top-level I/O ports for any design. Place and route tools that generate post-route VHDL models are usually hard-coded to use these data types. If you do not follow this convention, it is likely that your testbench will not "bind" to the design when post-route simulation is performed.

Testbench Design Overview

A VHDL testbench is just another VHDL entity. One that instantiates the design that you are creating and provides stimulus to it. A testbench can also verify that the correct outputs are generated by your design.

Broadly speaking, VHDL testbenches are either self-contained or non self-contained. Self-contained testbenches generate the entire stimulus using VHDL constructs, whereas non self-contained testbenches use VHDL textio functions to perform file input/output. Non self-contained testbenches read stimulus from a text file, apply it to the

inputs of the design, and then read the expected outputs of the design from a file, comparing them to the actual outputs.

The following section provides an example of a self-contained testbench for an 8-bit Adder.

Example Testbench for an 8-bit Adder

As an example of a fully self-contained testbench, consider the testbench shown below. This testbench is a simple design that performs unsigned addition on two 8-bit operands: din0 and din1, producing the output named dout.

```
library ieee;
use ieee.std_logic_1164.all;
library dataio;
use dataio.std_logic_ops.all;
entity testbench is
end testbench;
architecture behavior of testbench is
    component add8
      port(
        din0 : in std_logic_vector(7 downto 0);
        din1 : in std_logic_vector(7 downto 0);
        dout : out std_logic_vector(7 downto 0));
      end component;
      constant vector_period : time := 100 ns;
      signal din0 : std_logic_vector(7 downto 0) := (others => '0');
      signal din1 : std_logic_vector(7 downto 0) := (others => '0');
      signal dout : std_logic_vector(7 downto 0);
    begin
      uut: add8 port map(din0, din1, dout);
      p_stim: process
      begin
        for i in 0 to 255 loop
          for j in 0 to 255 loop
            if (i + j <= 255) then
              din0 <= to_vector(8, I);
              din1 <= to_vector(8, j);
              wait for vector_period;
              assert (dout = to_vector(8, i+j))
              report "Expected dout = "& integer'image(i+j)&
              " Actual dout = " &
```

```
            integer'image(to_integer(dout)) severity error;
          end if;
        end loop;
      end loop;
  end process;
end;
```

Looking at the entity declaration, you can tell you are looking a testbench, since this entity has no inputs and no outputs. In essence, a testbench always acts as the whole "outside world" to the design being tested.

The architecture declaration section declares a constant called vector_period, which is used to time the testbench. Explicit time values could have been spread throughout the testbench, but using constant(s) makes it much easier to modify the testbench later if the timing characteristics of the design change significantly. Also in the architecture declaration is a component declaration for the design and signal declarations for all input and output signals. Initial values are provided for the inputs.

It is good practice to provide initial values for signals. If you do not and then forget to make sure that the signal has a valid value at time 0, the signal defaults to the value 'U' (assuming std_ulogic or std_logic types). This in turn may cause one or more outputs of your design to become fixed at the value 'X'. Since testbenches can often grow to have more lines of code than the design you are testing, it is important to follow good coding practices.

In the architecture body of the testbench, the add8 design is instantiated and then the stimulus is provided using a single process. Note that VHDL allows you to create the loop variables *i* and *j* simply by using them. This represents a rare violation of VHDL style of strong type checking and self-documenting code. A subtle point to notice here is that *i* and *j* are created implicitly as unbounded integers, even though they are used for values in the range of 0 to 255 only. If the expected declarations for *i* and *j* had been used:

```
variable i, j : integer range 0 to 255;
```

the testbench would generate a fatal error during simulation as soon as the sum of the two values exceeded 255. This would occur during execution of the line:

```
if (i + j <= 255) then
```

which is intended to limit the range of input combinations applied to the design. Since the inputs and output of the design are std_logic_vectors, the conversion functions to_vector and to_integer are used to convert back and forth to integer values. These functions are in the package dataio.std_logic_ops.

An assertion is used to check the output of the adder. The assert statement does what it implies. It asserts that the given condition must be true. If the condition is not true, the text string that follows the report keyword is output in the console window of the simulator. A severity level is normally displayed along with the report string. You can use the severity level of your choice. Valid choices are note, warning, error, or fatal. VHDL simulators usually allow you to set the simulator to stop when an assertion at a certain severity level occurs. Integer'image is a construct that converts an integer value into a string representation. An example of what this would display if the assertion failed is:

```
** Error: Expected dout = 27 Actual dout = 31
  Time: 28000 ns Iteration: 2 Instance:/
```

Chapter 13

Synthesis

All design flows in Foundation Series ISE include synthesis. The synthesis stage creates optimized EDIF netlists used for placement and routing into Xilinx devices.

This chapter contains the following sections:

- "Overview"
- "Changing Synthesis Tools"
- "ABEL Synthesis (CPLDs Only)"
- "XST Synthesis"
- "FPGA Express Synthesis"

Overview

Three synthesis tools are available in Foundation Series ISE: ABEL, XST (Xilinx Synthesis Technology), and FPGA Express from Synopsys. The targeted device and intended sources (VHDL / Verilog / ABEL code or schematics) for the design are main considerations in choosing a synthesis tool for your project. If your project targets a Virtex, VirtexE, Virtex2, Spartan2, or any Xilinx CPLD device and does *not* mix VHDL and Verilog sources, you can use either the XST or FPGA Express tool. If you want to target any Xilinx XC4000 family, Spartan, or SpartanXL device or *mix* VHDL and Verilog code in your project, you must use FPGA Express.

Regardless of the synthesis tool, the overall synthesis procedure is the same. First, you select a source in the Source window and then select a synthesis process in the Process window.

If desired, you can allow the automake feature of Foundation Series ISE to handle all the synthesis for your design. Just select a source in

the Source window and then click any of the Synthesize process in the Process window. All the necessary synthesis processes are then automatically run to get the design source to the requested stage.

You may want to add timing constraints and view synthesis results to fine-tune your design for the targeted device before you implement it in that device. To do this, Foundation Series ISE includes support for setting synthesis process properties, using constraints files, and viewing synthesis reports. The available synthesis processes (listed under **Synthesize** in the Process window) and the process properties available for the Synthesize process vary based on the synthesis tool and device you select for your project. The "XST Synthesis" section describes the XST processes and the "FPGA Express Synthesis" section describes the FPGA Express processes.

For both tools, the Synthesize process performs the following tasks:

- Creates a VHDL or Verilog netlist for schematic sources
- Analyzes and Verifies source code
- Synthesizes logic from VHDL, Verilog, and ABEL source code and from schematic netlists to target a specific Xilinx device
- Generates a VHDL or Verilog netlist for functional simulation (FPGA Express only)
- Optimizes logic for speed / area, etc. as directed by design constraints
- Creates and exports an EDIF netlist optimized for the targeted device
- Analyzes and reports on timing for the design

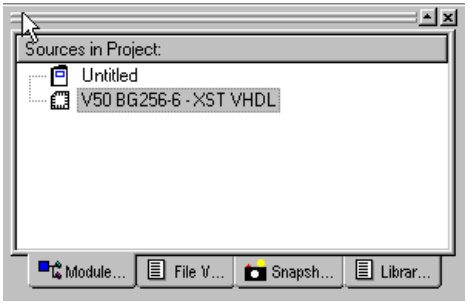
Changing Synthesis Tools

You select a synthesis tool when you create your project as described in "Selecting a Device and Synthesis Tool" section. At any point in the design process, you can change the synthesis tool designated for the project using the procedure described in this section. Before you change to a different synthesis tool, you need to be aware of the following consequences:

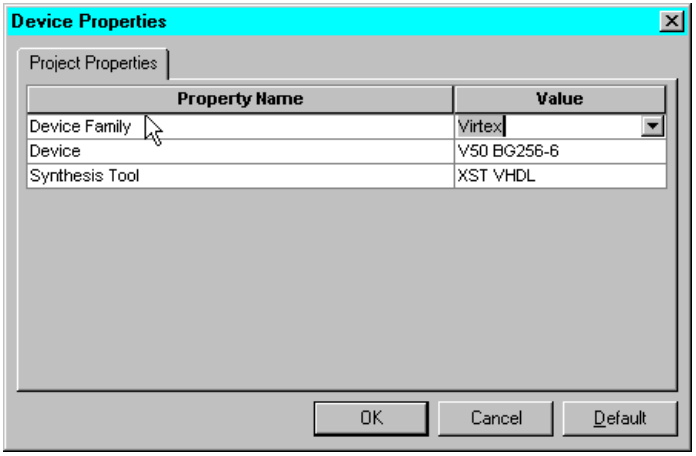
- You should archive your project or take a snapshot of it before you change synthesis tools if you want to retain the data generated by the initial synthesis tool. Refer to the "Snapshots" chapter for information on archives and snapshots.
- No automatic processing is done when a different synthesis tool is selected. You must re-run synthesis on your design to have the design processed with the new synthesis tool.
- Be aware that when you change from one synthesis tool to another, you may need to make other changes to your design. For example, if you change from XST VHDL to XST Verilog, all your VHDL source modules become invalid for the XST Verilog flow. Or, if you want to change from a Virtex device to a Spartan device and are currently using the XST synthesis tool. You would need to change to the FPGA Express tool because Spartan devices are not supported in XST.

Use the following procedure to change synthesis tools after you create your project:

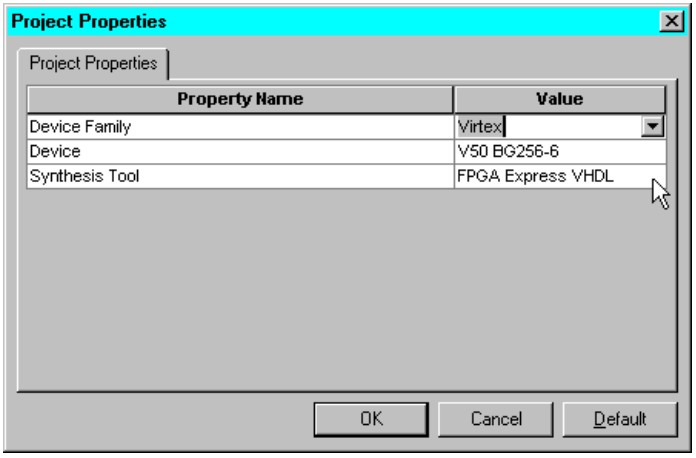
1. Click on the Device / Synthesis Tool line in the Source window to highlight it.



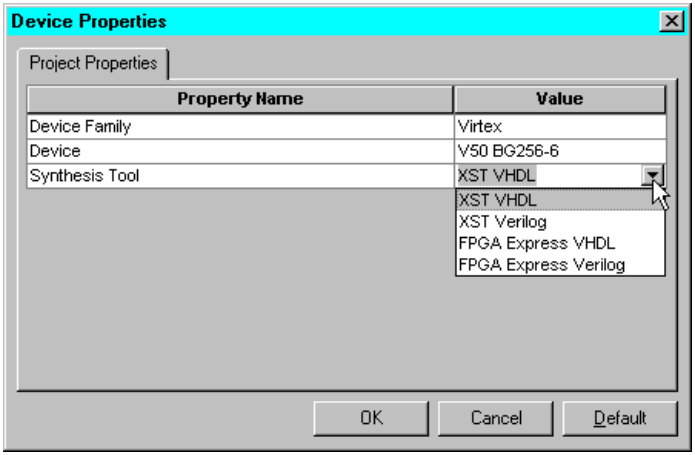
2. Select **Source** → **Properties** from the Project Navigator menu to display the Device Properties dialog box. (Or, you can double-click on the Device / Synthesis Tool line to display this dialog box.)



3. When the Device Properties dialog box appears, click in the right side of the Value field for the Synthesis Tool property to enable a pull-down menu to make the selection.



4. Select a synthesis tool from the Value pull-down menu.



The selection of a synthesis tool is closely linked to the Xilinx device family and sources (VHDL, Verilog, schematic, ABEL-HDL) you want to include in your project. For more information on the relationship between the device, synthesis tool, and design entry tools refer to “Selecting a Device and Synthesis Tool” section.

Schematics are supported in all tools. Schematics are first netlisted to HDL and then synthesized. The tools create an HDL functional model (netlist) of the schematic for synthesis as follows:

- ABEL, XST VHDL, and FPGA Express VHDL create VHDL functional models for schematics
- XST Verilog and FPGA Express Verilog create Verilog functional models for schematics

Note Both FPGA Express VHDL and FPGA Express Verilog support mixed VHDL and Verilog designs. *For FPGA Express, the designation “VHDL” or “Verilog” refers only to how any schematic netlists in the project are compiled.*

ABEL Synthesis (CPLDs Only)

ABEL synthesis applies only to ABEL-HDL modules in CPLD designs. The ABEL-XST and ABEL-BLIF synthesis tools are described in the following sections.

ABEL-XST

The ABEL-XST synthesis flow is provided as a higher-performance ABEL flow. The optimization and processing tools used in the ABEL-XST flow are generally more efficient, modern, and robust than the tools that have long been provided in the traditional ABEL-BLIF flow. Additionally, the ABEL-XST flow allows users greater control over the synthesis of their ABEL-HDL designs. This includes a special WYSIWYG feature designed to preserve the original logic equation structure in an ABEL design throughout implementation into the CPLD device.

Projects that use the ABEL-XST synthesis tool must target a CPLD device. They can include the following design entry sources:

- ABEL-HDL
- Schematics
- State Machines
- LogiBLOX Modules

ABEL-HDL and schematic sources are netlisted to VHDL and are synthesized using XST.

The following figure shows the Process window for an ABEL source in an ABEL-XST flow. You can right click on many of the processes to access a Process Properties dialog box containing the options you can set for the selected process. Within the Process Properties dialog box, you can use the **F1** help function to display detailed help information on the options.

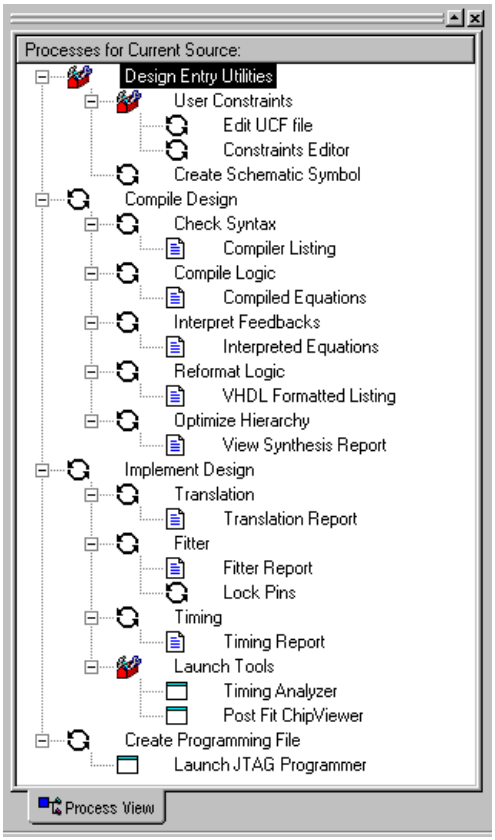


Figure 13-1 Process Window for an ABEL Source in the ABEL-XST Flow

ABEL-BLIF

The ABEL-BLIF flow provides access to traditional ABEL compilation tools intended to produce compatible results for existing ABEL designs. Projects that use the ABEL-BLIF synthesis tool must target a CPLD device. They can include the following design entry sources:

- ABEL-HDL
- Schematics
- State Machines

- LogiBLOX Modules

ABEL-HDL sources are translated to BLIF before synthesis where they are optimized and directly netlisted for implementation without further synthesis.

Schematic sources are netlisted to VHDL and synthesized using XST.

The following figure shows the Process window for an ABEL source in an ABEL-BLIF flow. You can right click on many of the processes to access a Process Properties dialog box containing the options you can set for the selected process. Within the Process Properties dialog box, you can use the **F1** help function to display detailed help information on the options.

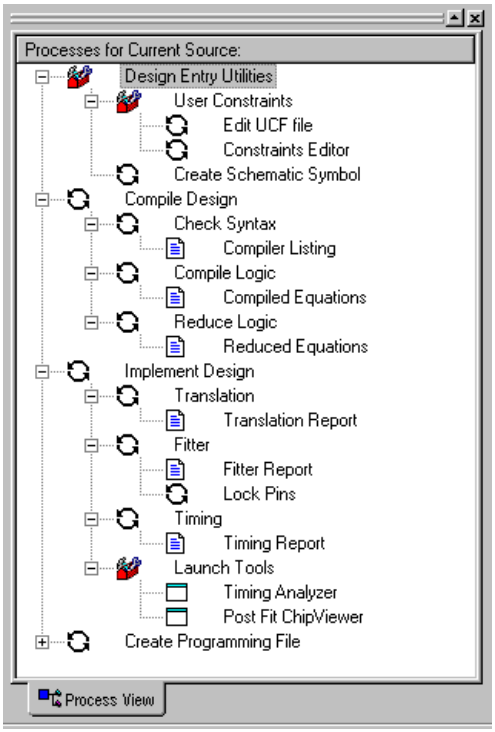


Figure 13-2 Process Window for an ABEL Source in the ABEL-BLIF Flow

XST Synthesis

The Xilinx Synthesis Technology (XST) tool synthesizes HDL designs to create EDIF netlists optimized for the targeted Xilinx device. You can control the synthesizing of your design by setting process properties and using constraints files prior to synthesis.

XST supports two flows: XST VHDL and XST Verilog.

XST VHDL

Projects that use the XST VHDL synthesis flow must contain only VHDL sources. Schematics are netlisted to VHDL prior to synthesis. The Synthesize processes for XST VHDL are shown in Figure 13-3.

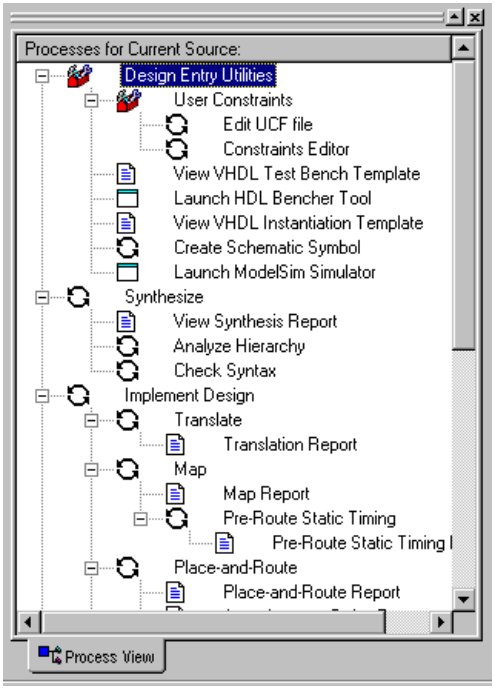


Figure 13-3 XST VHDL Synthesis Processes

XST Verilog

Projects that use the XST Verilog synthesis flow must contain only Verilog sources. Schematics are netlisted to Verilog prior to synthesis. The Synthesize processes are shown in Figure 13-4.

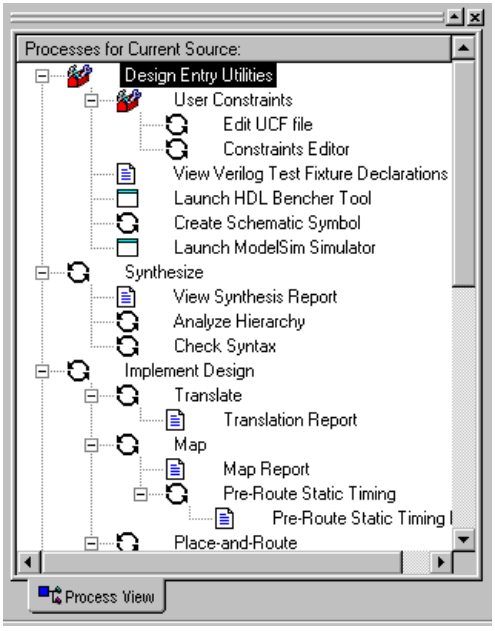


Figure 13-4 XST Verilog Synthesis Processes

Selecting a Top-Level Source for Synthesis

To select a top-level source for synthesis, simply click on the desired source in the Source window. That source and all sources below it in the design hierarchy are then processed when a synthesis process is invoked in the Process window.

To synthesize the entire design, select the top-level design source. Running synthesis on lower-level design sources is useful for checking whether the source is synthesizable and for checking inferred macros. You can also get timing and size estimations from running these partial designs.

XST Synthesis Processes

You can invoke a synthesis process within the Project Navigator as follows:

1. Select a top-level source in the Sources window. Refer to "Selecting a Top-Level Source for Synthesis" section for information on top-level sources.
2. Double-click a synthesis process. (**`Synthesize`** or any process or report listed under it) in the Processes window.

Prior to creating the optimized EDIF netlist for the use by the implementation tools, you can optionally do the following to check / verify your design:

- Check the syntax of the HDL code.

  Double-click **`Check Syntax`** under the Synthesis process in the Process window. Errors are reported in the Transcript window.

- Analyze the HDL code for correct syntax and verify that it follows the correct HDL policy for the selected synthesis tool. Verify the port connections between files.

  Double-click **`Analyze`** under the Synthesis process in the Process window. Errors are reported in the Transcript window.

- Set synthesis options that specify design-specific constraints to produce the desired performance in the targeted device.

  If the default constraints are not sufficient for your requirements, you can set performance constraints, attributes, and optimization controls before synthesizing the design. Refer to "Setting XST Synthesis Options" section for information. You may need to try different values multiple times before you find the right value to meet your design requirements.

Viewing Synthesis Results

A summary of the synthesis results is printed in the Project Navigator's transcript window as synthesis is performed. You can view a report on the synthesis results in the Reports Viewer using the following procedure:

1. Select a source in the Source window.

2. Double-click on **`View Synthesis Report`** in the Process window.

If synthesis had not been previously run or if the previous results are out-of-date, the necessary processes are automatically run before the report is opened in the Reports Viewer.

The XST Synthesis Report includes sections that list the current synthesis property settings, processing status, the HDL synthesis report, and a Timing Report.

Constraining the Design

Before you start synthesis to optimize your design for the targeted device, you can optionally set performance constraints, attributes, and optimization controls. Entering your design requirements as constraints can improve the placement and routing results of your design.

With XST you can control synthesis in the following ways:

- Using the Synthesis process properties described in the "Setting XST Synthesis Options" section.
- Entering XST-specific constraints and attributes directly into the HDL code. Refer the *XST User Guide* for information on XST-specific constraints.
- Entering constraints in a UCF file. Refer to the *XST User Guide* for information on XST and UCF constraints.
- Entering constraints in the XST constraints file. Refer to the *XST User Guide* for detailed information on the XST constraints file.

Changing Speed Grades

A typical method to get your design to meet timing requirements is to try various speed grades for the targeted device. For FPGAs you can change the speed grade prior to XST synthesis in either of the following ways:

- Right-click on the Device / Synthesis Flow line in the Source window. Select **`Properties`** from the pull-down menu. When the Device Properties dialog box appears, use the pull-down menu in the Device field to change the speed grade for the

device. (The last two characters of the device name identify the speed grade.)

- Select a source in the Source window. Right-click on **Synthesize** in the Process window. When the synthesis Process Properties dialog box appears, select the **Xilinx Specific Options** tab. Enter a new speed grade in the Value field for the **Speed Grade for Timing Analysis** option.

Note You can only make speed grade selections / changes for CPLD devices after synthesis. For CPLDs, the speed grade is set as an implementation property. (Select a source in the Source window. Right-click on **Implement Design** in the Process window. Enter the speed grade in the **Speed Grade** field of the **Design** tab.)

To determine the devices and corresponding speed grades installed on your PC, do the following:

1. Select the Device / Synthesis Tool line in the Source window.
2. Select **Display Device Information** in the Process window.
3. The Report Viewer opens with the list of installed devices.

Setting XST Synthesis Options

You can set synthesis process options to control XST synthesis. To access the XST synthesis process properties, do the following:

1. Select the desired top-level source in the Source window.
2. Right-click on **Synthesize** (or **Optimize Hierarchy** with ABEL-XST for ABEL-HDL sources) in the Process window.
3. Select **Properties** to display the Process Properties dialog box.

The synthesis Process Properties dialog box for XST contains three tabs: Synthesis Options, HDL Options, Xilinx Specific Options. You can use the **F1** help function to display detailed descriptions of the options on each tab.

Note The options set in the synthesis Process Properties dialog box are *global* properties for your design. If you want to set source-specific properties, you need to enter them in an XST constraints file. Refer to the *XST User Guide* for information on XST constraints files.

Synthesis Options

The available Synthesis Options depend on the targeted device and whether VHDL or Verilog is used. An example list of options (for a Virtex XST VHDL project) is shown in the following figure.

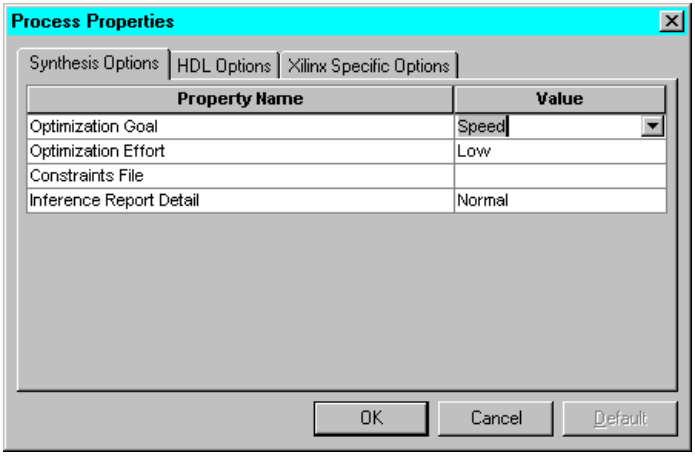


Figure 13-5 XST Synthesis Properties (Virtex)

You can set the following *global* synthesis options for XST processing:

- Optimization Goal

  Value: Speed (default) / Area

  The option defines the synthesis optimization strategy. By default, XST optimizations are speed-oriented.

- Optimization Effort

  Value: Low (default) / High

  This option defines the synthesis optimization effort level. Allowed values are Low (normal optimization) and High (higher optimization).

- (CPLDs only) Flatten Hierarchy

  Value: Disabled (default) / Enabled

  For CPLDs, to obtain a completely flattened design, enable the following options:

  - Flatten Hierarchy enabled (Synthesis Options)
  - Macro Preserve disabled (Xilinx Specific Options)
  - XOR Preserve disabled (Xilinx Specific Options)

- Constraints File path

  Value: filename and path

  You can optionally enter the name of a constraints file (an XST constraints file) containing XST-specific constraints. These constraints are used by XST processing and can include module / entity-specific constraints.

- Inference Report Detail

  Value: Normal (default) / Low / Verbose).

  This option controls the amount of detail reported on inferred macros in the Synthesis Report.

- (Verilog Only) Case Implementation Style

  Value: Full (default) / Parallel / Full-Parallel

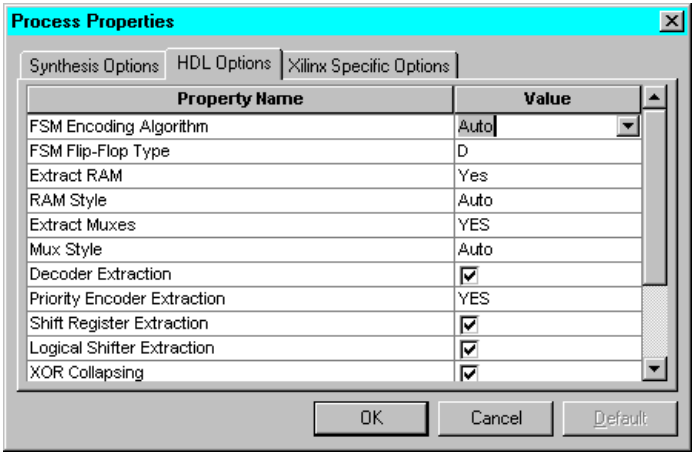
  This option sets the case implementation style directive for synthesis.

Refer to the *XST User Guide* for detailed information on and examples of these properties.

HDL Options

The available HDL Options depend on the targeted device and whether VHDL or Verilog is used. An example list of options (for a Virtex XST VHDL project) is shown in the following figure.

Note A check mark in a Value check box enables the property. A blank check box indicates that the property is disabled. Click in the Value box to enable / disable the property.



You can set the following HDL Options for XST synthesis:

- FSM Encoding Algorithm

  Values: Auto (default) / One-hot / Compact / Sequential / Gray / Johnson / User / none)

  This option selects the finite state machine (FSM) encoding technique to be used. The default is **`Auto`**, meaning that the best coding technique is automatically selected for each individual state machine. **`User`** indicates a user-customized technique. Refer to Table 13-1 for descriptions of the standard FSM encoding techniques.

Table 13-1 State Encoding Techniques

Technique	Description
One-Hot	One-hot encoding associates one code bit, and also one flip-flop, to each state. At a given clock cycle during operation, one and only state variable is asserted. Only two state variables toggle during a transition between two states. One-hot encoding is very appropriate with most FPGA targets where a large number of flip-flops are available. It is also a good alternative when trying to optimize speed or to reduce power dissipation.
Gray	Gray encoding guarantees that only one state variable switches between two consecutive states. It is appropriate for controllers exhibiting long paths without branching. In addition, this coding technique minimizes hazards and glitches. Very good results can be obtained when implementing the state register with T or JK flip-flops.
Compact	Compact encoding consists of minimizing the number of state variables and flip-flops. This technique is based on hypercube immersion. Compact encoding is appropriate when trying to optimize area.
Johnson	Like Gray, Johnson encoding shows benefits with state machines containing long paths with no branching.
Sequential	Sequential encoding consists of identifying long paths and applying successive radix two codes to the states on these paths. Next state equations are minimized.

- FSM flip-Flop Type

  Value: D (default) / T / JK

  This option specifies whether a state register in a FSM should be implemented as a D, T, or JK flip-flop.

- Extract RAM

  Value: Yes (default) / No

  The Extract RAM property allows you to enable (Yes) or disable (No) RAM macro inference.

- RAM Style

  Value: Auto (default) / Distributed / Block

  The RAM Style property controls the way the macrogenerator implements the RAM macros. When set to Auto (the default), XST looks for the best implementation for each considered macro. When set to Distributed, RAM macros are specified as Distributed RAM. When set to Block, RAM macros are specified as Block RAM.

- Extract Muxes

  Value: Enabled (default) / Disabled

  This option enables or disables multiplexer macro inference. When enabled, for each identified multiplexer description, XST creates a macro or optimizes it with the rest of the logic, based on internal decision rules.

- Mux Style

  Value: Auto (default) / MUXF / MUXCY)

  This option controls the way the macrogenerator implements the multiplexer macros. The default value is Auto, meaning that XST looks for the best implementation for each considered macro. Available implementation styles for the Virtex and Spartan2 series are based on either MUXF5/F6 resources or MUXCY resources.

- Decoder Extraction

  Value: Enabled (default) / Disabled

  This option enables or disables decoder macro inference.

- Priority Encoder Extraction

  Value: Enabled (default) / Disabled

  This option enables or disables priority encoder macro inference. When enabled, for each identified priority encoder description, based on internal decision rules, XST creates a macro or optimizes it with the rest of the logic.

- Shift Register Extraction

  Value: Enabled (default) / Disabled

  This option enables or disables shift register macro inference.

- Logical Shifter Extraction

  Value: Enabled (default) / Disabled

  This option enables or disables logical shifter macro inference.

- XOR Collapsing

  Value: Enabled (default) / Disabled

  This option controls whether cascaded XORs should be collapsed into a single XOR.

- Resource Sharing

  Value: Enabled (default) / Disabled

  This option enables or disables resource sharing of arithmetic operators.

- Complex Clock Extraction

  Value: Enabled (default) / Disabled

  Sequential macro inference in XST generates macros with clock enable functionality whenever possible. This option instructs or prevents the inference engine to not only consider basic clock enable templates, but also look for less obvious descriptions where the clock enable can be used.

- Resolution Style

  Value: WIRE-MS (default) / WIRE-OR / WIRE-AND

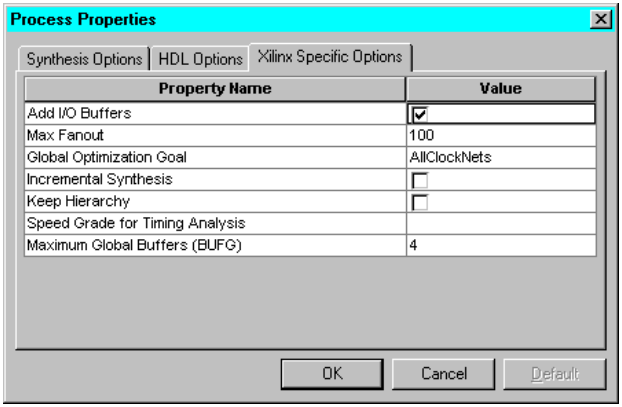
  The resolution style option controls how multi-source situations not protected by tri-state logic are handled. The WIRE-MS value is assumed by default and instructs XST to exit with an error condition whenever such a multisource situation is found. With the WIRE_OR and WIRE_AND resolution styles, all detected situations are replaced by respectively OR-based logic or AND_based logic, and synthesis continues.

Refer to the *XST User Guide* for detailed information on and examples of these properties.

Xilinx Specific Options (FPGAs)

The available Xilinx Specific Options depend on the targeted device and whether VHDL or Verilog is used. An example list of options for FPGA devices (for a Virtex XST VHDL project) is shown in the following figure.

Note A check mark in a Value check box enables the property. A blank check box indicates that the property is disabled. Click in the Value box to enable / disable the property.



You can set the following *global* HDL Options for XST synthesis:

- Add I/O Buffers

  Value: Enabled (default) / Disabled

  When this option is enabled, XST automatically inserts Input/Output buffers into the design for I/Os that do not have manually instantiated I/O buffers. This option is useful to synthesize a part of a design to be instantiated later on.

- Max Fanout

  Value: 100 (default)

  This option can be used to limit the fanout of nets in the whole design by specifying the maximum fanout.

- Global Optimization Goal

  Value: Clock Frequency (default) / Inpad to Outpad / Inpad to Setup / Clock to Outpad / All to Setup / All to Outpad / Maxdelay)

XST optimizes the region of the design specified in this option. By default, global optimization is tuned for clock frequency minimization.

- Incremental Synthesis

  Value: Disabled (default) / Enabled

  If the Incremental Synthesis option is enabled, each entity / module of the design is synthesized to a single, separate netlist.

- Keep Hierarchy

  Value: Disabled (default) / Enabled

  XST may automatically flatten the design to get better results by optimizing across entity / module boundaries. You can enable the Keep Hierarchy option so that the generated netlist is hierarchical and respects the hierarchy and interface of any entity / module of your design.

- Speed Grade for Timing Analysis

  Value: same as targeted device (default)

  For the timing analysis, XST take into account the speed grade. By specifying a different speed grade, XST will perform a different optimization due to different timing information.

- (Advanced Option) Maximum Global Buffers (BUFG)

  Value: 4 (default)

  XST automatically inserts BUFGs for clock signals. If you want XST to use less than the maximum number of BUFGs available, then specify a lower number. This option is useful if the design to be synthesized is not complete and if the missing part contains BUFGs.

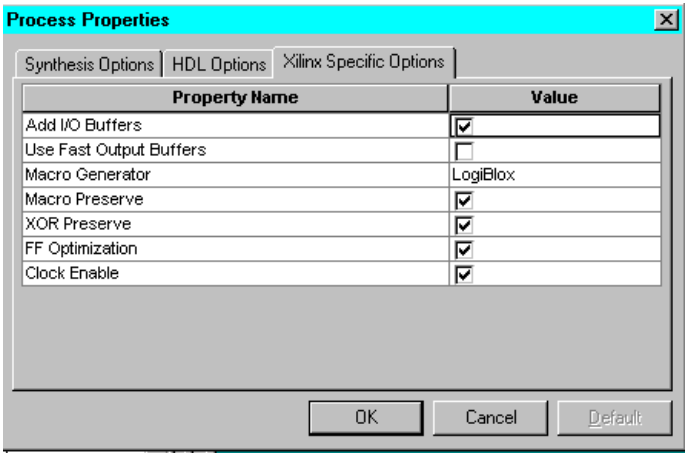
  Note You must have **Advanced** options selected for the Property display level in the **Edit** → **Preferences** → **Processes** dialog box to access the Maximum Global Buffers option.

Refer to the *XST User Guide* for detailed information on and examples of these properties.

Xilinx Specific Options (CPLDs)

The available Xilinx Specific Options depend on the targeted device and whether VHDL or Verilog is used. An example list of options for CPLD devices (for a XC9500XL XST VHDL project) is shown in the following figure.

Note A check mark in a Value box enables the property. A blank Value box indicates that the property is disabled. Click in the Value box to enable / disable the property.



You can set the following *global* HDL Options for XST synthesis:

- Add I/O Buffers

  Value: Enabled (default) / Disabled

  When this option is enabled, XST automatically inserts Input/Output buffers into the design for I/Os that do not have manually instantiated I/O buffers. This option is useful to synthesize a part of a design to be instantiated later on.

- Use Fast Output Buffers

  Value: Disabled (default) / Enabled

  The Use Fast Output Buffers property allows you to enable or disable the use of fast output buffers.

- Macro Generator

  Value: LogiBlox (default) / Macro+

  A macro inferred by the HDL synthesizer is passed to a CPLD low-level synthesizer which calls a macrogenerator to create its implementation. Two macrogenerators are available:

  - Macro+ (XST's internal macrogenerator)
  - LogiBlox macrogenerator

  A macro submitted by the HDL synthesizer may be accepted or rejected by the CPLD synthesizer. An accepted macro becomes a hierarchical block in the final netlist, its logic being generated by Macro+ or by the CPLD fitter (LogiBlox macro).

  The Macro Generator option is tightly related to the Macro Preserve option: the macros are generated only if Macro Preserve is yes. Otherwise, the macros are replaced by equivalent logic units generated by HDL synthesizer.

- Macro Preserve

  Value: Enabled (default) / Disabled

  This option is useful for making the macro handling independent of design hierarchy processing (see Flatten Hierarchy, a Synthesis Option). So you can merge all hierarchical blocks in the top module, but you can still keep the macros as hierarchical modules. When the Macro Preserve option is enabled, macros are preserved and generated by Macro+ or LogiBlox. When disabled, macros are rejected and generated by the HDL synthesizer.

  Depending on the Flatten Hierarchy value, a rejected macro becomes a hierarchical block (Flatten Hierarchy is disabled) or is merged in the design logic (Flatten Hierarchy is enabled). Please note that very small macros (2-bit adders, 4-bit multiplexers) are always merged, independently of the Macro Preserve or Flatten Hierarchy options.

- XOR Preserve

  Value: Enabled (default) / Disabled

  The XORs inferred by HDL synthesis are also considered as macro blocks in the CPLD flow. They are processed separately to give more flexibility for the use of device macrocells XOR gates.

Therefore, you can decide to flatten the design (enable Flatten Hierarchy and disable Macro Preserve) but to preserve the XORs.

Preserving XORs has a great impact on reducing design complexity. Preserving the XORs, generally, gives better results; the number of PTerms is lower. The preserved XORs appear in the EDIF netlist as LogiBlox XOR macros and are expanded by the CPLD fitter.

When the XOR Preserve option is disabled, XOR macros are merged with surrounded logic. This is useful to obtain completely flat netlists. Applying the global optimization on a completely flat design can improve the design fitting.

Note Disabling this option does not guarantee the elimination of the XOR operator from the EDIF netlist. During the netlist generation, the netlist mapper tries to recognize and infer XOR gates in order to decrease the logic complexity. This process is independent of the XOR preservation done by HDL synthesis and is guided only by the goal of complexity reduction.

- FF Optimization

  Value: Enabled (default) / Disabled

  When this option is enabled, XST removes the following flip-flops to increase fitting success:

  - Equivalent flip-flops, having the same inputs and control signals
  - Flip-flops with constant input/control signals

  Note The flip-flop optimization algorithm is time consuming. If fast processing is desired, this option should be disabled.

- (XC9500XL, XC9500XV Only) Clock Enable

  Value: Enabled (default) / Disabled

  The macrocells of the XC9500XL and XC9500XV families support the Clock Enable signal. When the Clock Enable option is enabled, the CPLD synthesizer implements the use of the clock enable signal of the device. When disabled, the clock enable function is implemented through equivalent logic.

  Keeping or not keeping the clock enable signal depends on the design logic. Sometimes, when the clock enable is the result of a

Boolean expression, disabling this option may improve the fitting result because the input data of the flip-flop is simplified when it is merged with the clock enable expression.

Refer to the *XST User Guide* for detailed information on and examples of these properties.

Detailed Information on XST

Detailed information on using XST can be found in the following sources:

- *XST User Guide*

  The *XST User Guide* contains complete information on XST support for Xilinx devices, HDL languages, and design constraints. It also explains how to use various design optimization and coding techniques when creating HDL designs for use with XST.

- Online help

  You can access XST's online help from the Project Navigator Help menu as follows.

  - Select **Help** → **Foundation Series ISE Help Contents** from the Project Navigator.
  - Then select **XST** (under Synthesis) in the Xilinx Foundation Series ISE On-Line Help System menu.

- *Synthesis and Simulation Design Guide*

  This manual provides a general overview of designing Field Programmable Gate Arrays (FPGAs) with Hardware Description Languages (HDLs). It includes design hints for the novice HDL user, as well as for the experienced user who is designing FPGAs for the first time.

Note The *XST User Guide* and *Synthesis and Simulation Design Guide* are available in the DocSan-searchable online book collection as well as a .pdf file. Both formats are available on the Documentation CD included with this release. Or, you can select **Help** → **Online Books** to access the online books.

FPGA Express Synthesis

FPGA Express is a logic-synthesis optimization tool from Synopsys. It includes functions to analyze source code, synthesize, optimize, generate EDIF netlists, and analyze timing. FPGA Express has two flows in ISE: FPGA Express VHDL and FPGA Express Verilog.

FPGA Express VHDL

Projects that use the FPGA Express VHDL flow can contain VHDL or Verilog sources. Schematics are netlisted to VHDL prior to synthesis. The FPGA Express processes are shown in Figure 13-6.

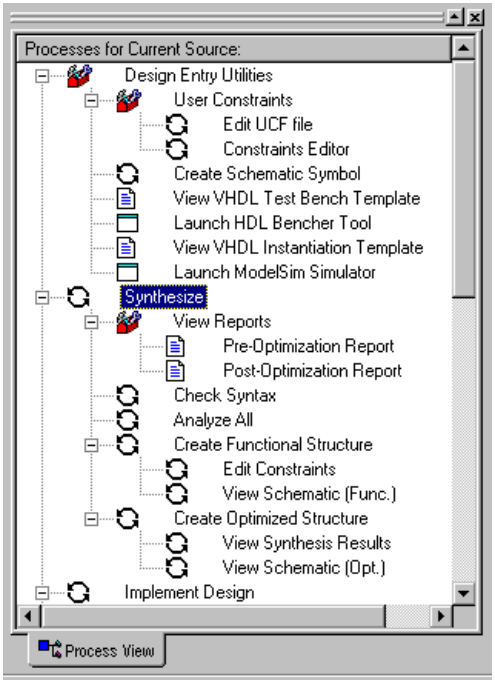


Figure 13-6 FPGA Express Synthesis Processes

FPGA Express Verilog

Projects that use the FPGA Express Verilog flow can contain VHDL or Verilog sources. Schematics are netlisted to Verilog prior to synthesis.

The synthesize processes for FPGA Express Verilog are the same as those shown in Figure 13-6 for FPGA Express VHDL.

Selecting a Top-Level Source

To select a top-level source, simply click on the desired source in the Source window. That source and all sources below it in the design hierarchy are then synthesized when a synthesis process is invoked in the Process window.

To synthesize the entire design, select the top-level design source. Running synthesis on lower-level design sources is useful for checking whether the source is synthesizable and for checking inferred macros. You can also get timing and size estimations from running these partial designs.

FPGA Express Synthesis Processes

You can invoke an FPGA Express synthesis process within the Project Navigator as follows:

1. Select a top-level module in the Sources window. Refer to "Selecting a Top-Level Source for Synthesis" section for information on top sources.
2. Double-click a synthesis process. (**`Synthesize`** or any process listed under it) in the Processes window.

Prior to creating the optimized EDIF netlist for the use by the implementation tools, you can optionally do the following to check that your design meets your project requirements:

- Check the syntax of the HDL code.

  Click **`Check Syntax`** under the Synthesis process in the Process window.

- Analyze the HDL code for correct syntax and verify that it follows the correct HDL policy for the selected synthesis tool.

  Click **`Analyze`** under the Synthesis process in the Process window.

The following processes (Create Functional Structure and Create Optimized Structure) makeup the overall synthesis process. You can run each process manually rather than automatically as part of the synthesize process if desired.

- Create a functional structure

  The Create Functional Structure performs the design elaboration. To run the Create Functional Structure process, do the following.

  - Select the design module or file you have updated.
  - Right click on **`Create Functional Structure`** in the Process window.
  - Click **Run**. (Or, you can double-click on the Create Functional Structure Process to initiate the Run process.)

  You can view the new functional structure with the View Schematic (Func) process.

- Create an optimized structure

  The Create Optimized Structure process performs architecture-specific optimization on the design. To run the Create Optimized Structure process, do the following.

  - Select the design module or file you have updated.
  - Right click on **`Create Optimized Structure`** in the Process window.
  - Click **Run**. (Or, you can double-click on the Create Optimized Structure Process to initiate the Run process.)

  You can view the optimized structure with the View Schematic (Opt) process.

- Set synthesis options that specify design-specific constraints to produce the desired performance in the targeted device.

  If the default constraints are not sufficient for your requirements, you can set performance constraints, attributes, and optimization controls before synthesizing the design. Refer to the “Setting FPGA Express Synthesis Options” section for information.

- In the FPGA Express flow, you can also optionally set timing constraints, create a functional model to test the constraints, and view the results of the specified constraints. Refer to the

"Constraining the Design" sectionfor more information on this option.

Constraining the Design

Before you start synthesis to optimize your design for the targeted device, you can optionally set performance constraints, attributes, and optimization controls. Entering your design requirements as constraints can improve the placement and routing results of your design.

Setting Constraints Prior to Synthesis

Global synthesis optimization goals and other constraints can be set in the Synthesis Process Properties dialog box.

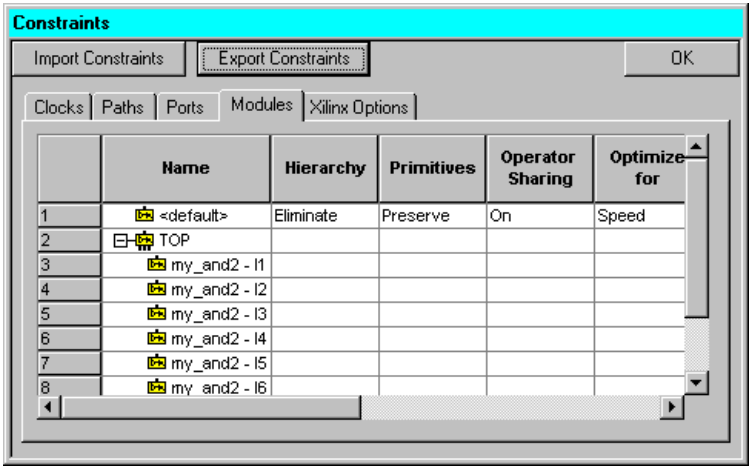
You can set performance constraints and attributes to guide the optimization process on a module-by-module basis. This means that you have the ability to, for example, optimize certain modules of your design for speed and some for area. In addition, an effort level for the optimization engine can be set to either high or low.

To set constraints in FPGA Express on a module-by-module basis, do the following:

1. Select the desired top-level module.
2. Select **`Edit Constraints`** in the Synthesize section of the Process window.
3. The Express Constraints Editor opens. Set the desired constraints in the Express Constraints Editor GUI.

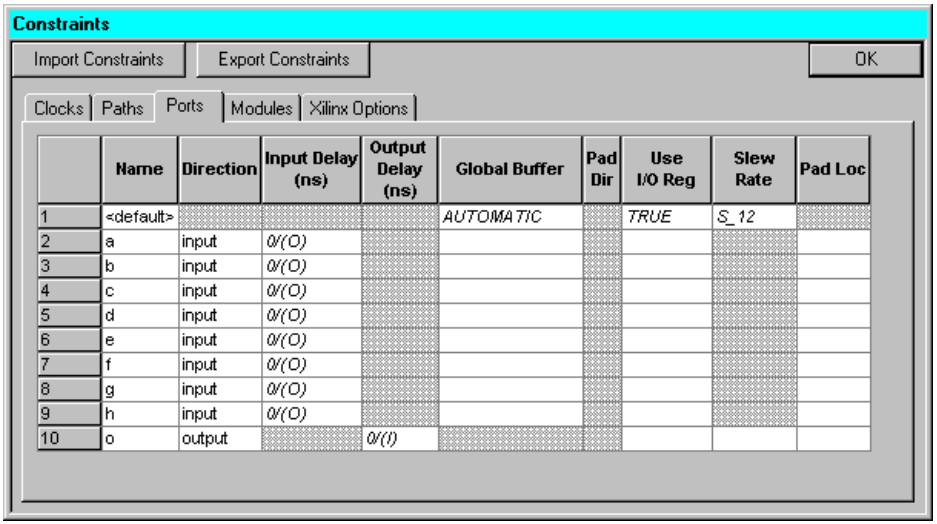
The Express Constraints Editor window contains five different tabs. The spreadsheets and dialog boxes on the tabs are specific to the target architecture.

Optimization goals are set for individual modules in the Module tab of the Express Constraints Editor. The Module tab is shown in the following figure.



The following three tabs contain constraints that can be applied to the design prior to synthesis: Clock, Paths, and Ports. These constraints are passed to the Place and Route tools via the design netlist and NCF constraints file.

- The Clocks tab allows you to specify overall speeds for the clocks in a design.
- The Paths tab allows you precise control of point-to-point timing in a design.
- The Ports tab allows OFFSETS, pull ups / pull downs, and pin locations to be specified in a design.



	Name	Direction	Input Delay (ns)	Output Delay (ns)	Global Buffer	Pad Dir	Use I/O Reg	Slew Rate	Pad Loc
1	<default>				AUTOMATIC		TRUE	S_12	
2	a	input	0/(O)						
3	b	input	0/(O)						
4	c	input	0/(O)						
5	d	input	0/(O)						
6	e	input	0/(O)						
7	f	input	0/(O)						
8	g	input	0/(O)						
9	h	input	0/(O)						
10	o	output		0/(I)					

The timing constraints specified in the Express Constraints Editor tabs are translated into FROM:TO or PERIOD timespecs and placed in an NCF file. Following is an example:

```
TIMESPEC TS_CLK = PERIOD “CLK” 20 ns HIGH 10;
```

UCF File Constraints

Currently, FPGA Express cannot apply all Xilinx constraints that may be entered in a UCF file. It can apply the following constraints:

- PERIOD
- FROM:TO timespecs which use FFS, LATCHES, and PADS
- Pin location constraints
- Slew rate
- TNM_NET
- PULLUP / PULLDOWN
- OFFSET:IN:BEFORE
- OFFSET:OUT:AFTER

Express cannot apply the constraints listed below. To enter these constraints, use the Xilinx Constraints Editor:

- TPSYNC
- TPTHRU
- TIG
- user-RLOCs, RLOC_ORIGIN, RLOC_RANGE
- non-I/O LOCs
- KEEP
- U_SET, H_SET, HU_SET
- user-BLKNM and user-HBLKNM
- PROHIBIT

Express creates its own timegroups by grouping logic with common clocks and clock enables. In addition, you can form user-created timing subgroups by right clicking on an existing timing path and choosing New Sub Path in the Express Constraints Editor's Paths tab.

Viewing Synthesis Results

A summary of the synthesis results is printed in the Project Navigator's transcript window as synthesis is performed. You can check the synthesis results prior to implementing the design by viewing the Post-Optimization Report (select **`Post-Optimization Report`** in the Synthesize section of the Process window). FPGA Express supplies two other methods of verifying your design prior to implementation: the FPGA Express Time Tracker and the Schematic Viewer. All these tools enable you to view synthesis results prior to implementation. This allows you to adjust constraints or try different devices, for example, multiple times until you find the right combination to meet your design's timing requirements.

Report Viewer

In FPGA Express flows, you can view synthesis results at two different points in synthesis processing:

- At the pre-optimized (RTL) stage

  To view a report (the Synopsys Chip-Top report) on the pre-optimized design, select the top-level module / source in the Source window and then double click on **`Pre-Optimization Report`** under View Reports in the Process window.

- At the optimized stage

  Double-click on **`Post-Optimization Report`** in the Process window to view the report (the Synopsys Chip-Top Optimized report) in the Project Navigator's Report Viewer.

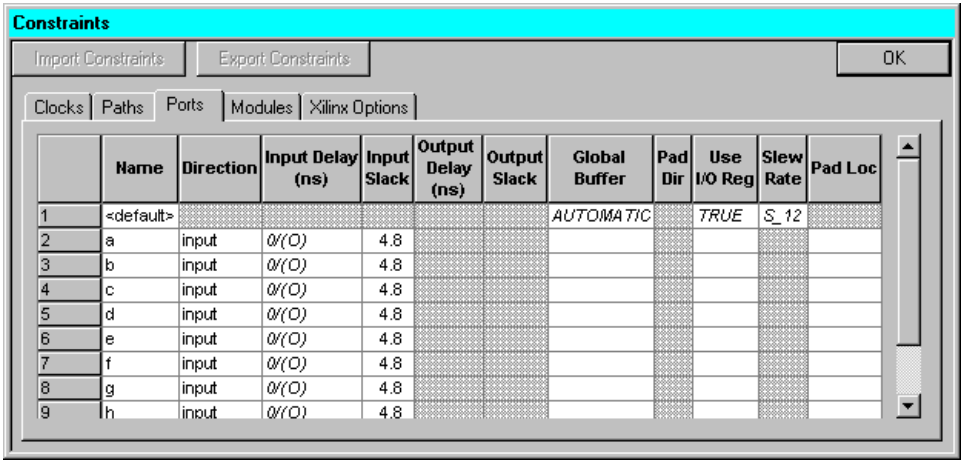
If synthesis had not been previously run or if the previous results are out-of-date, the necessary processes are automatically run before the report is opened in the Reports Viewer.

FPGA Express Time Tracker

You can check the post-synthesis timing results in the Express Constraints Editor, as follows:

1. Select the desired top-level source module.
2. Select **`View Synthesis Results`** in the Synthesize section of the Process window.
3. The Express Time Tracker opens. The Time Tracker displays the estimated results for the design.

The timing data is displayed in the same format as used to set the constraints. An example Time Tracker window for the Ports tab is shown in the following figure.



	Name	Direction	Input Delay (ns)	Input Slack	Output Delay (ns)	Output Slack	Global Buffer	Pad Dir	Use I/O Reg	Slew Rate	Pad Loc
1	<default>						AUTOMATIC		TRUE	S_12	
2	a	input	0/(O)	4.8							
3	b	input	0/(O)	4.8							
4	c	input	0/(O)	4.8							
5	d	input	0/(O)	4.8							
6	e	input	0/(O)	4.8							
7	f	input	0/(O)	4.8							
8	g	input	0/(O)	4.8							
9	h	input	0/(O)	4.8							

Schematic Viewer

FPGA Express provides a Schematic Viewer tool that allows you to view and analyze your design graphically.

To open an RTL view of the design, select a source module / entity in the Source window and then click **`View Schematic (Func.)`** in the Process window. Viewing the pre-optimized design allows you to evaluate the connectivity of a circuit and to get a feel for how FPGA Express interpreted the RTL being processed.

To evaluate how the design was mapped to the targeted device, you can view a gate-level representation. To open a gate-level (mapped / optimized) view of the design, select a source module / entity in the Source window and then click **`View Schematic (Opt.)`** in the Process window. The Schematic Viewer can be used with the Time Tracker to view the timing results graphically.

Changing Speed Grades

A typical method to get your design to meet timing requirements is to try various speed grades for the targeted device. For FPGAs, you can change the speed grade prior to FPGA Express synthesis as follows:

1. Right-click on the Device / Synthesis Flow line in the Source window.
2. Select **`Properties`** from the pull-down menu.

3. When the Device Properties dialog box appears, use the pull-down menu in the Device field to change the speed grade for the device. (The last two characters of the device name identify the speed grade.)

Note You can only make speed grade selections / changes for CPLD devices after synthesis. For CPLDs, the speed grade is set as an implementation property. (Select a source in the Source window. Right-click on **`Implement Design`** in the Process window. Enter the speed grade in the **`Speed Grade`** field of the **`Design`** tab.)

To determine the devices and corresponding speed grades installed on your PC, do the following:

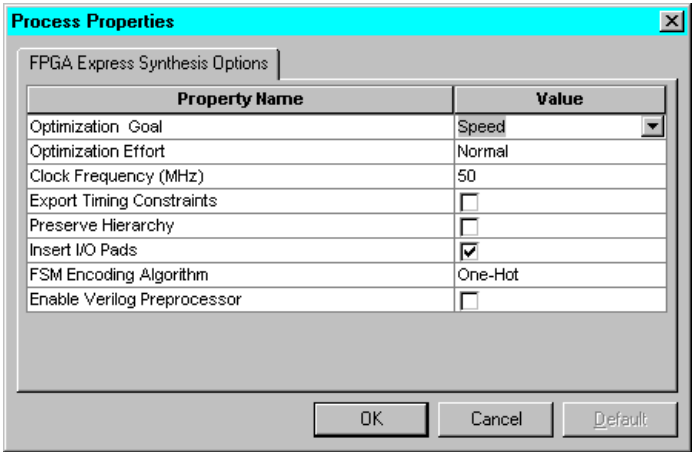
1. Select the Device / Synthesis Tool line in the Source window.
2. Select **`Display Device Information`** in the Process window.
3. The Report Viewer opens with the list of installed devices.

Setting FPGA Express Synthesis Options

You can control FPGA Express synthesis by setting synthesis process properties. To set synthesis properties, do the following:

1. Select the desired top-level source in the Source window.
2. Right-click on **`Synthesize`** in the Process window
3. Select **`Properties`** to display the synthesis Process Properties dialog box.

Note A check mark in a Value check box enables the property. A blank check box indicates that the property is disabled. Click in the Value box to enable / disable the property.



You can set the following HDL Options for FPGA Express synthesis:

- Optimization Goal

  Value: Speed (default) / Area

  “Speed” specifies that the design be optimized for faster speed. Area specifies that the design be optimized for a smaller area.

- Optimization Effort

  Value: Low (default) / High

  “Low” selects the quickest mapping method. “High” selects longer compilation resulting in better design mapping; mapping proceeds until all strategies have been tried.

- Clock Frequency (MHz)

  Value: 50 MHz (default)

  This property specifies the desired general clock frequency for a design implementation.

- Export Timing Constraints

  Value: Disabled (default) / Enabled

  Enable this option to export timing constraints to an NCF file when exporting a netlist.

- Preserve Hierarchy

  Value: Disabled (default) / Enabled

  Enable this option to preserve design hierarchy and output hierarchical netlists.

- Insert I/O pads

  Value: Enabled (default) / Disabled

  When enabled, specifies that I/O pads should be inserted. If this option is disabled, global buffers and global set/reset signals are not inferred or implemented and timing constraints are not exported with the netlist file.

- FSM Encoding Algorithm

  Value: Auto (default) / one-hot / compact / sequential / Gray / Johnson / user / none

  This option selects the finite state machine (FSM) encoding technique to be used. The default is auto, meaning that the best coding technique is automatically selected for each individual state machine. Refer to Table 13-1 for a description of the various techniques.

- Enable Verilog Preprocessor

  Value: Disabled (default) / Enabled

  This property controls whether the Verilog preprocessor is run or not.

Detailed Information on FPGA Express

Detailed information on using FPGA Express can be found in the following sources:

- Online help

  You can access FPGA Express's online help from the Project Navigator Help menu as follows. This help also includes a tutorial on using FPGA Express.

  - Select **`Help`** → **`Foundation Series ISE Help Contents`** from the Project Navigator.

- Then select **FPGA Express** (under Synthesis) in the Xilinx Foundation Series ISE On-Line Help System menu.

- *Synopsys VHDL Reference Guide*

  The *Synopsys VHDL Reference Guide* describes how to use FPGA Express to translate and optimize a VHDL description into an internal gate-level equivalent. (This book is available in the online book collection as a PDF file only.)

- *Synopsys Verilog Reference Guide*

  The *Synopsys Verilog Reference Guide* describes how to use FPGA Express to translate and optimize a Verilog description into an internal gate-level equivalent. (This book is available in the online book collection as a PDF file only.)

- *Synthesis and Simulation Design Guide.*

  This manual provides a general overview of designing Field Programmable Gate Arrays (FPGAs) with Hardware Description Languages (HDLs). It includes design hints for the novice HDL user, as well as for the experienced user who is designing FPGAs for the first time.

Note The books mentioned in this section are available on the Documentation CD included with this release. Or, you can select **Help** → **Online Books** to access the online books.

Chapter 14

Implementing the Design

After you have created 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 in order to achieve your design objectives.

This chapter contains the following sections:

Using the Process Window to Implement the Design

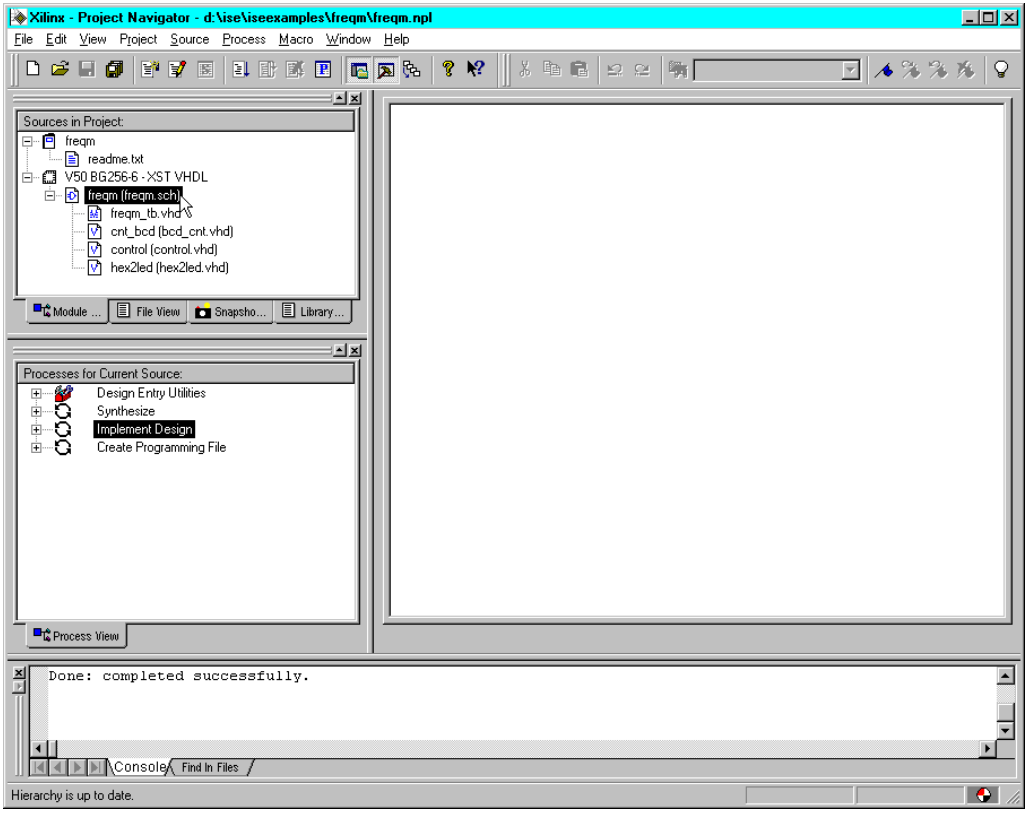
Checking the implementation of your design in the targeted device before you implement it is not a required step. For large designs, however, this step can help you to find problems and correct them or to fine tune the implementation to meet your design requirements. For instance, you can set implementation process options to reduce area, increase speed, and vary placement effort. Or, you can try targeting various devices to determine the most suitable fit for your design. If a particular device proves to be too large or too slow for your needs, you can select a smaller or faster target device from a different family.

The Implement Design processes for your design depend on whether your design is for an FPGA (see Figure 14-3) or CPLD device (see Figure 14-18). The synthesis tool selection does not affect the implementation flow. After design entry, you can use the Implement Design section in the Process window for a selected source to perform the following operations on that source:

- Implementation of the design for a specific target device
- Report generation showing the status of the design
- Timing analysis for design verification
- Export of the design for timing simulation and programming

You can implement your design completely or request that only the necessary processing be done to produce a specific implementation report that you want to check before continuing with the implementation. Use the following procedure to implement your design.

1. Click on a source file (an HDL file or schematic) in the Source window representing the design that you want to implement in the device (shown by the arrow in the following figure). That source and all sources in the design hierarchy below that source 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. See the “FPGA Implementation Options” section or the “CPLD Implementation Options” section for information on Setting Implementation Properties.

   Note Implementation process properties affect *all* sources in the project. They can *not* be set for (or associated with) only certain sources.

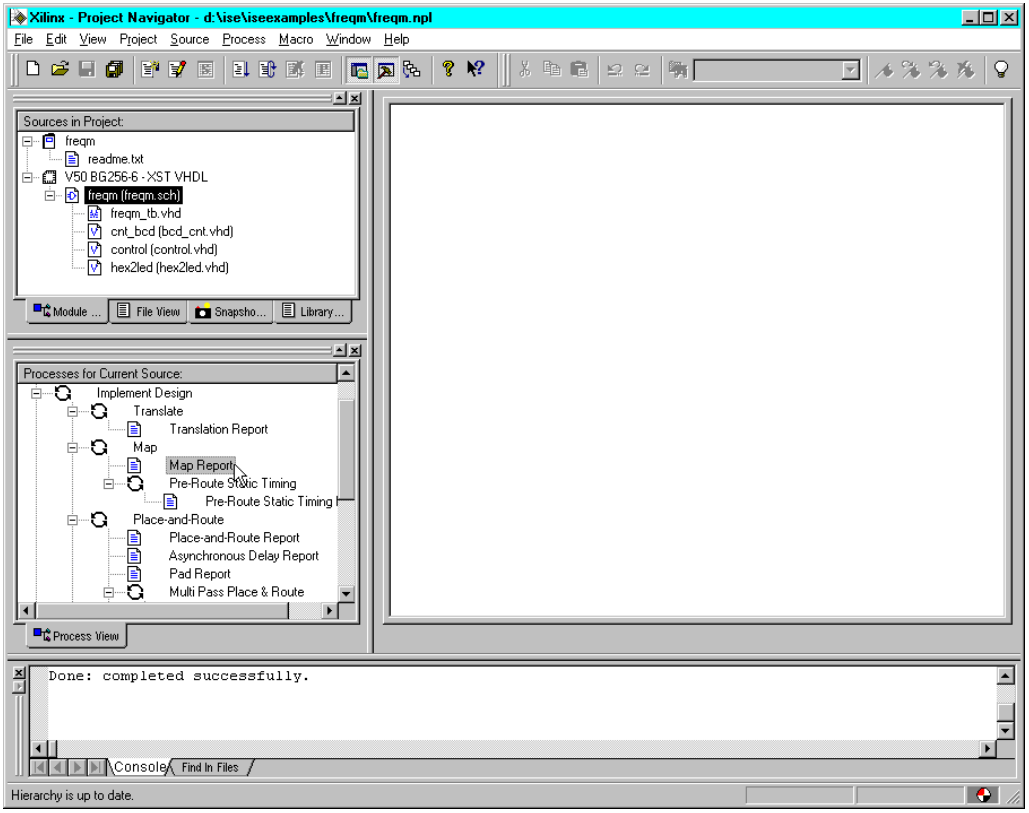
3. Use the Implement Design processes in the Process window to select the implementation processing you want performed on the selected source (from that source down through the design hierarchy). You can implement your design completely or request that only necessary processing be done to produce a specific implementation report.

- Complete implementation

  For complete implementation processing, double click **`Implement Design`** in the Process 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. At the completion of the processing, an icon appears to indicate the results. A green check mark indicates the design implemented without errors.

- Partial implementation to run a process or produce a report

  For example, if you want to view a report on the design only through a certain implementation stage, double click on the name of a report listed under the various processes in the Implement Design section of the Process window (**`Map Report`**, for example). You may need to click the "+" box beside the Implement Design process to expand the Implement Design process list as shown in the following figure. If the report does not currently exist, the necessary processing is done to produce the report for you to view. If the report exists, it is opened immediately.



- Specialized processing

  Click on 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 tools 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/Warnings

As a source is implemented, the processes being run and their status are indicated in the Transcript window. Refer to the "Error Navigation from the Transcript Window" section of the "Project Navigator" chapter for information on navigating to errors or warnings that appear in the Transcript window.

The processing status is also indicated by icons beside the process or report name in the Process window. At the completion of the processing, 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 "successfully processed." A red X indicates that errors were detected preventing successful completion of the process. A yellow ! (exclamation point) indicates a warning.

Saving Implementation Results

Two functions are available to preserve a specific version of the implementation. You can archive a version of the project as a .zip file. You can take a snapshot to preserve a project version for later reference and easy reactivation. The Snapshot View tab in the Source window lists the snapshots taken of the project. From the Snapshot View tab you can view a hierarchical listing of the files in a snapshot. You can also select reports in a snapshot for viewing. Refer to the "Snapshots" chapter for detailed information on snapshots.

Deleting Implementation Results

You can delete the implementation results by selecting **`Project`** → **`Delete Implementation Data`** from the Project Navigator menu. This removes all the files produced for all implementation processing done on the design. The files are deleted from the project directory.

Changing Devices

You may want to change device families to get the desired 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, double-click on the device / synthesis tool line in the Source window to display the Project Properties dialog box. Select a new device or device family from the Project Properties dialog box.

If you select a new device family, you will notice that the implementation results (and synthesis results) become "out-of-date." Select **`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. Refer to the "Creating a Project" 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, and I/O pin assignment for your design.

To generate and/or view a report, select a design source in the Source window and then double-click on the name of the desired report in the Process window Implement Design section. If the report currently exists (from previous processing), it is immediately opened in the ISE Report Viewer. If the report does not currently exist, ISE runs the necessary processes to produce an up-to-date report and then displays it in the ISE Report Viewer. If the existing report is out-of-date (no green check mark beside it), you can click on the report name and then select **`Process`** → **`Run`** (or **`Process`** → **`Rerun All`**) to update it before viewing it.

The reports produced depend on whether your design targets a CPLD or FPGA device.

ISE Report Viewer

The ISE Report Viewer is a separate window from the Project Navigator windows. When you select a report from the Implement Design section of the Process window, the report opens in the ISE Report Viewer. Figure 14-1shows an example of the Report Viewer window.

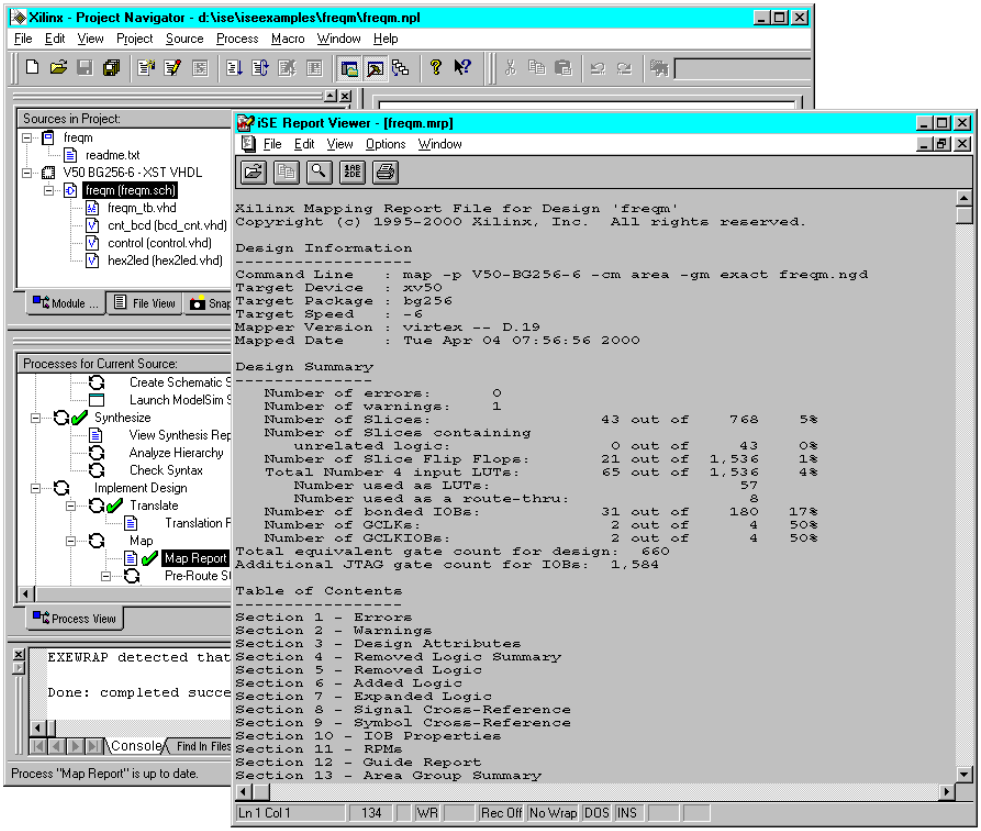
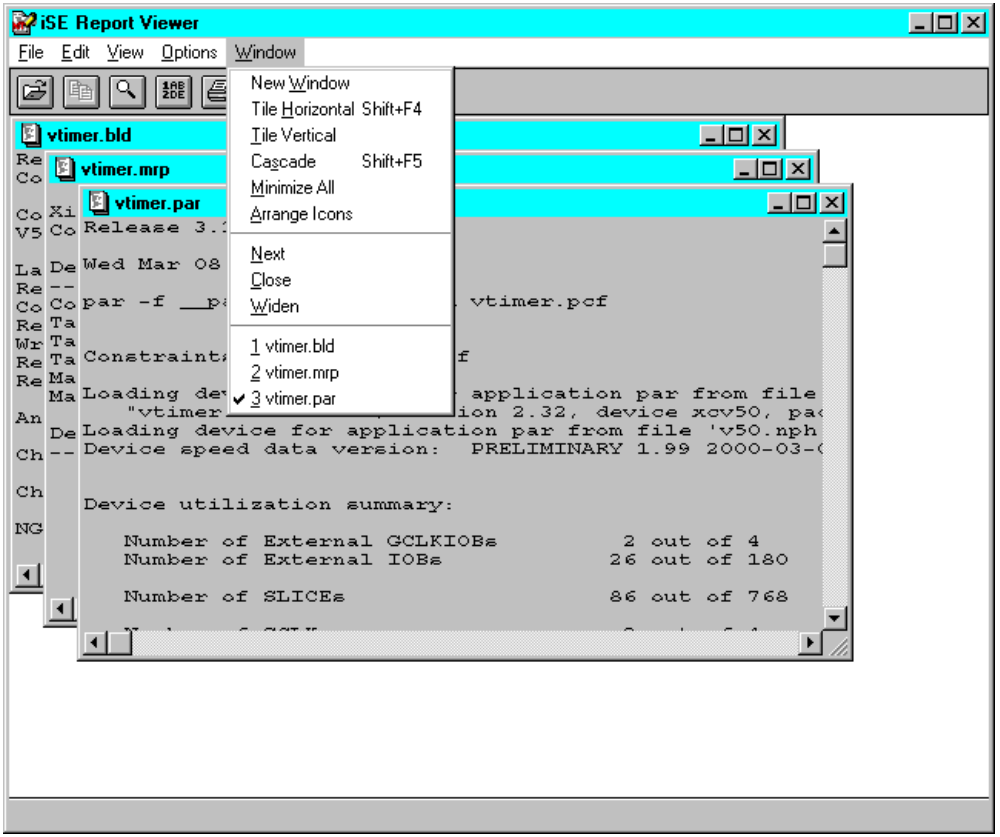


Figure 14-1 ISE Report Viewer Window

Initially, the report files are maximized when they are opened in the Reports Viewer. You can use the Tile or Cascade selections on the Window menu if you want to view two or more reports at once as shown in the following figure.



Reports displayed in the Reports Viewer are read-only. You can print the report or save it under another name, if desired. Print options included line numbering, line wrapping, page headers, and font selections.

The Report Viewer Edit menu contains search functions to search the current report. The View menu allows you to customize the display of the toolbar and status bar. You can set window and file modes in the Options menu as well as the screen font and keymappings for the Report Viewer.

You can view other files is the Report Viewer window by selecting **`View`** from the ISE Report Viewer menu and then browsing to a file. All files opened in the Report Viewer are opened read-only.

Report Descriptions

For descriptions of each of the reports listed in the Implement Design section of the Process window, refer to the "FPGA Implementation Reports" section and "CPLD Implementation Reports" section.

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 Process window provides access to the mechanisms you can use to enter or modify constraints on the implementation of your design.

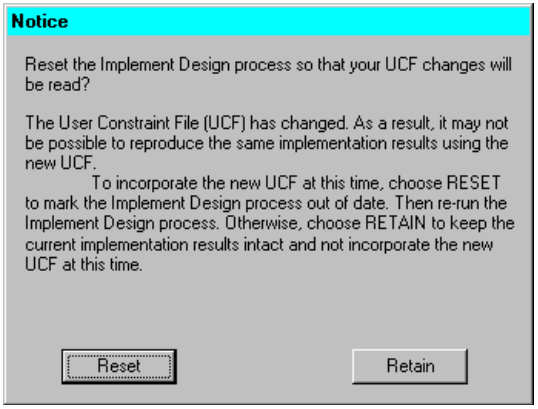
Editing the UCF File

The User Constraints File (UCF) is an ASCII file specifying constraints on the logical design. You can 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, blank 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. If you want to create or use a different UCF file, you can identify the UCF file to use for your project in the Translate Options tab of the implementation Process Properties for FPGA designs (see the "FPGA Implementation Options" section). For CPLDs, the UCF file for the project is specified in the Design tab (see the "CPLD Implementation Options" section).

To edit the current UCF file for the project, select a source in the Source window and then double-click **`Edit UCF File`** in the Design Entry Utilities section of the Process window. The current UCF file is immediately opened in Notepad (or your usual text editor) for viewing, printing, or modification. Refer to the "Attributes, Constraints, and Carry Logic" chapter of the *Libraries Guide* for information on constraints that can be entered in the UCF file.

Whenever the UCF file is modified and saved, the following Notice window appears to notify you that the implementation processes are not automatically updated when the UCF is changed.



You must indicate (by clicking **Reset**) that you want to have the implementation processing rerun with the new design source netlist and UCF file.

Accessing the Constraints Editor

You can use the Constraints Editor program 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 Constraints Editor are automatically added into the User Constraints File (UCF).

To access the Constraints Editor, select a source in the Source window and then double-click Constraints Editor in the Implement Design section of the Process window. The Constraints Editor opens with the design loaded. An example of the Constraints Editor initial window is shown in the following figure.

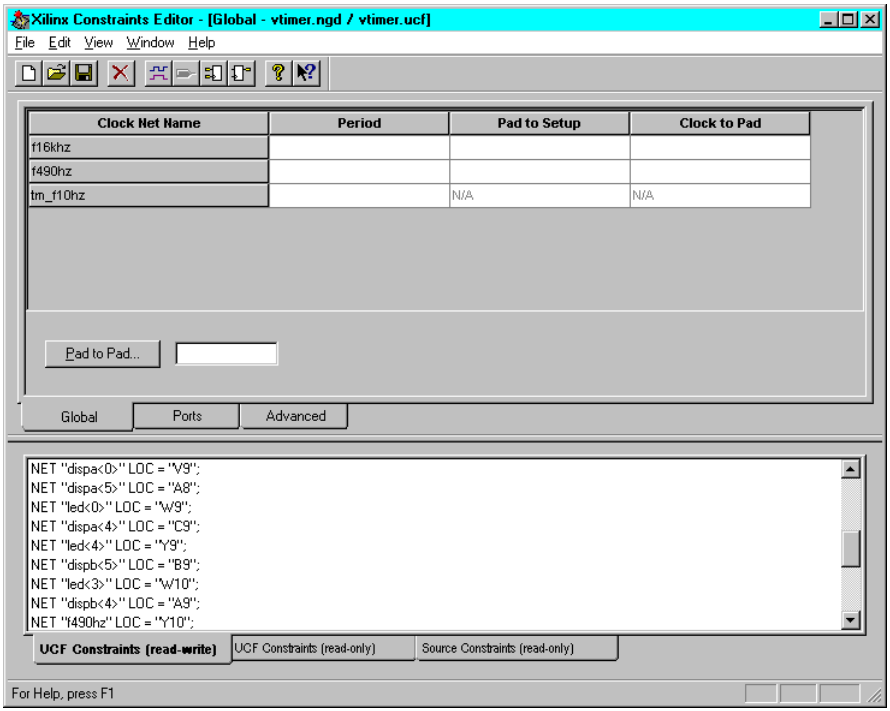


Figure 14-2 Xilinx Constraints Editor

The UCF file is changed when constraints are added or edited through the 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.

Refer to the Constraints Editor online help or to the *Constraints Editor User Guide*, an online book for detailed information on using the Constraints Editor.

Accessing the Chip Viewer (CPLDs)

You can access the ChipViewer before implementation and after implementation. Before running the Fitter, you can use the ChipViewer to control pin assignments for implementation. To access the ChipViewer for pin assignment, select the top-level design source in the Source window and then double-click on **Pin Assignment ChipViewer** in the User Constraint section under Design Entry Util-

ities. The ChipViewer opens with the complete netlisted source design loaded. The Pin Assignment ChipViewer process is only available if you have a specific CPLD device and package selected for your project.

After implementation is complete, you can use the ChipViewer to examine the physical mapping resulting from the Fitter. To access the ChipViewer to examine Fitter results, select your top-level design source and double-click on **Post Fit ChipViewer** in the Launch Tools section under Implement Design. The ChipViewer opens with the completely mapped design loaded.

More information on using the CPLD ChipViewer is available in that tool's online help (**Tools** → **Implementation** → **CPLD ChipViewer** → **Help**) or from the Foundation Series ISE Help menu accessed by **Help** → **Foundation ISE Help Contents** → **Advanced Tools** → **ChipViewer**.

FPGA Implementation Flow

The following figure shows the implementation portion of the Process window for a design that targets an FPGA device. The implementation processes are under “Implement Design.”

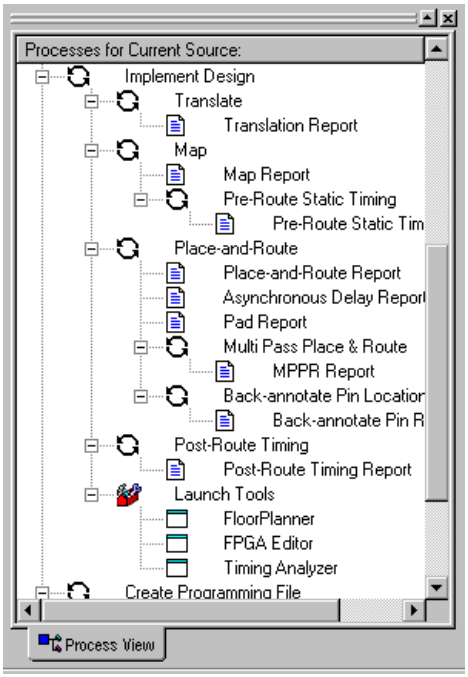


Figure 14-3 FPGA implement Design Processes

When you select a source and then click **`Implement Design`**, the necessary processing to implement the design in the targeted device is performed. Default implementation processing properties are used unless you modify them as described in the “FPGA Implementation Options” section.

Xilinx implementation tools are used to process your design. If you are familiar with the Xilinx Alliance product, you will notice that the command line entries that appear in the Transcript window as each function is run correspond to the Alliance development system commands.

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. 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.

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 types of files are *input files* for the translation process.

- Design file 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 schematic editor tool 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 file containing the implementation of a module in the design. LogiBLOX creates an NGC file to define each module.

The following types of files 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. Refer to the "Viewing Implementation Reports" section for information on this report.

For a complete description of Translate process, refer to the "NGDBuild" chapter of 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 (*top_source_name*.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 Options tab of the implementation Process Properties dialog box (see the“FPGA Implementation Options” section) prior to implementing the design.

  Note Virtex, VirtexE, Virtex2, and Spartan2 do not support guide files.

- (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 Options tab of the implementation Process Properties dialog box (see the“FPGA Implementation Options” section) prior to 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 (*top_source_name*.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 (*top_source_name*.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. Refer to the Xilinx *Libraries Guide* for a description of this file.

- Native Generic Map File (NGM file)

  An NGM file (*top_source_name*.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 (*top_source_name*.mrp) contains information about the Map run and its subprocesses. Refer to the "Viewing Implementation Reports" section for information on this report.

For a detailed description of the MAP process, see the *Development System Reference Guide*.

Pre-Route Static Timing (Optional)

The Pre-Route Static Timing process (the Xilinx TRACE program) creates timing simulation data from which you can ascertain if the timing requirements and functionality of your design are correct before the place and route process is implemented.

The following file are *input files* for the Pre-Route Static Timing process:

- A mapped design file (NCD file)

  The NCD file (*top_source_name*.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 (*top_source_name*.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. Refer to 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 pre-route static timing process outputs the Pre-Route Static Timing Report. The Pre-Route Static Timing Report (*top_source_name*.tw1) contains information about how well the timing constraints for the design have been met. Refer to the "Viewing Implementation Reports" section for information on this report.

For a complete description of the timing processes, refer to the "TRACE" chapter of 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 place and route. 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 as follows.

- *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 (*top_source_name*.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 (*top_source_name*.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. Refer to 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 implementation 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 (*top_source_name*.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 (*top_source_name*.par) includes summary information of all placement and routing iterations. Refer to the "Viewing Implementation Reports" section for information on this report.

- Asynchronous Delay Report (DLY file)

  The Delay File (*top_source_name*.dly) contains delay information for each net in the design. Refer to the "Viewing Implementation Reports" section for information on this report.

- Pad Report (PAD file)

  The PAD Report (*top_source_name*.pad) contains I/O pin assignments. Refer to the "Viewing Implementation Reports" section for information on this report.

- 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 "PAR—Place and Route" chapter in the *Development System Reference Guide*.

Post-Route Timing (Optional)

The Post-Route Timing process (the Xilinx TRACE program) can be run after Place and Route to create timing simulation data from which you can ascertain 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 (*top_source_name*.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 (*top_source_name*.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. Refer to 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 (*top_source_name* .twr). The Post-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. Refer to the "Post-Route Timing Report" section for information on this report.

For a complete description of the pre-Route Static Timing process, refer to the "TRACE" chapter of 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.

On average, design speed from an arbitrary multiple pass place and route will vary plus or minus 5 to 10% from the median design speed across all cost tables. About 1/3 of the cost tables will result in speeds within 5% of the median, 1/3 in the 5 to 10% range, and 1/3 in the 10 to 15% range. When comparing performance from the absolute worst cost table to the absolute best, a spread of 25 to 30% 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% 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, your design should be modified to remove one or more logic levels.

To run Multi Pass Place & Route on an FPGA design, select a top-level source in the Source window and then double-click on **`Multi Pass Place & Route`** in the Process 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 described in the "Multi-Pass Place & Route Options" section.

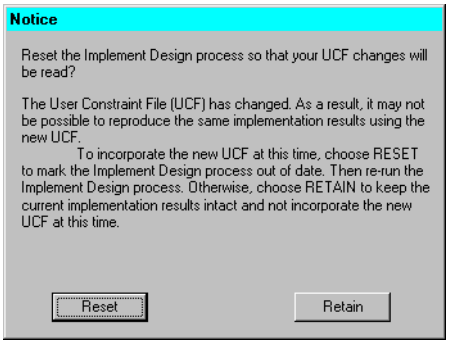
Refer to the "MPPR Report" section for information on the report produced by this Multi Pass Place and Route process.

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 (*top_source_name*.ncd).

When you are ready to commit the pinout of a design, select the top-level source file in the Source window and then double-click the **`Backannotate Pin Locs`** process for FPGA designs. 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 (*top_source_name*.ucf or the .ucf file specified in the implementation Process Properties). This pinout will then be applied to all subsequent design implementations that you run.

The Notice informational window shown in the following 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 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 on **Backannotate Pin Locs Report** in the Process window to generate and view this report. The Backannotate Pin Locs Report (*top_source_name*.lck for FPGAs) is displayed in the ISE Report Viewer.

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. Prior to 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” and 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.

FPGA Implementation Reports

This section contains descriptions of the reports available for information on the implementation processing of your FPGA design.

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

Map Report

The Map Report (*top_source_name*.mrp) contains warning and error messages detailing logic optimization and problems in mapping logic to physical resources. The report lists the following information:

- Removed logic. Sourceless and loadless signals can cause a whole chain of logic to be removed. Each deleted element is listed with progressive indentation, so the origins of removed logic sections are easily identifiable; their deletion statements are not indented.
- Logic that has been added or expanded to optimize speed.
- Component information. The Design Summary section lists the number and percentage of used CLBs, IOBs, flip-flops, and latches. It also lists occurrences of architecturally-specific resources like global buffers and boundary scan logic.

To get the complete (detailed) Map Report, you need to enable the "Generate Detailed MAP Report" option in the Map Options tab of the implementation Process Properties dialog box (see the "FPGA Implementation Options" section).

Note The Map Report can be very large. To find information, use key word searches. To quickly locate major sections, search for the string '---' , because each section heading is underlined with dashes.

Pre-Route Static Timing Report

The Pre-Route Static Timing Report (*top_source_name*_preroute. twr) 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.

Pre-route static timing reports can be very useful in evaluating timing performance. Although route delays are not accounted for, the logic delays can provide valuable information about the design.

If logic delays account for a significant portion (> 50%) of the total allowable delay of a path, the path may not be able to meet your timing requirements when routing delays are added.

Routing delays typically account for 40% to 60% of the total path delays. By identifying problem paths, you can mitigate potential problems before investing time in place and route. You can redesign the logic paths to use less levels of logic, tag the paths for specialized routing resources, move to a faster device, or allocate more time for the path.

If logic-only-delays account for much less (<35%) than the total allowable delay for a path or timing constraint, then the place-and-route software can use very low placement effort levels. In these cases, reducing effort levels allow you to decrease runtimes while still meeting performance requirements.

The timing report lists statistics on the design, any detected timing errors, and a number of warning conditions. Timing errors indicate absolute or relative timing constraint violations. These include the following.

- Path delay errors - where the path delay exceeds the maximum delay constraint for a path.
- Net delay errors - where a net connection delay exceeds the maximum delay constraint for the net.
- Offset errors - where either the delay offset between an external clock and its associated data-in pin is insufficient to meet the internal logic's timing requirements or the delay offset between an external clock and its associated data-out pin exceeds the external logic's timing requirements.

- Net skew errors - where skew between net connections exceeds the maximum skew constraint for the net.
- Timing errors may require design modifications, running PAR, or both.
- Warnings point out potential problems such as circuit cycles or a constraint that does not define any paths.

Different types of reports are available. Refer to the "Pre-Route Static Timing Options" section for information on the report types and other options you can for this report.

Place and Route Report

The Place and Route Report (*top_source_name*.par) contains the following information.

- The overall placer score which measures the "goodness" of the placement. Lower is better. The score is strongly dependent on the nature of the design and the physical part that is being targeted, so meaningful score comparisons can only be made between iterations of the same design targeted for the same part.
- The Number of Signals Not Completely Routed should be zero for a completely implemented design. If non-zero, you may be able to improve results by using re-entrant routing or the multi-pass place and route flow.
- The timing summary at the end of the report details the design's asynchronous delays.

Pad Report

The Pad Report (*top_source_name*.pad) lists the design's pinout in three ways.

- Signals are referenced according to pad numbers.
- Pad numbers are referenced according to signal names.
- PCF file constraints are listed. This section of the Pad Report can be cut and pasted into the .pcf file after the SCHEMATIC END; statement to preserve the pinout for future design iterations.

Asynchronous Delay Report

The Asynchronous Delay Report (*top_source_name*.dly) lists all nets in the design and the delays of all loads on the net. The worst net delays are listed at the top of the report.

MPPR Report

The Multi-pass Place and Route Report (MPPR) Report (*top_source_name*.mppr) is displayed in the ISE Report Viewer. The MPPR Report lists the design scores and timing scores for the design.

Post-Route Timing Report

The Post-Route Timing Report (*top_source_name*.twr) is an evaluation of the design's timing constraints, clock frequencies, and path delays. Both the logic and routing delays are incorporated to generate this report.

Post-Route timing reports incorporate all delays to provide a comprehensive timing summary. If a placed and routed design has met all of your timing constraints, then you can proceed by creating programming data and downloading to a device. On the other hand, if you identify problems in the timing reports, you can try fixing the problems by increasing the placer effort level, using re-entrant routing, or using multi-pass place and route. You can also redesign the logic paths to use fewer levels of logic, tag the paths for specialized routing resources, move to a faster device, or allocate more time for the paths.

FPGA Implementation Options

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. The options you can set for each of these processes are described in this section.

Accessing the Implementation Process Properties Dialog Box.

Use the following procedure to access the implementation Process Properties dialog box to modify implementation options.

1. Click on a design source file in the Source window for a project that targets an FPGA device.

   Note Implementation properties are set for the whole design. Not for the selected source file only.

2. Right click on **`Implement Design`** in the Process window.
3. Select **`Properties`** from the pull-down menu that appears.
4. The Process Properties dialog box for the Implement Design processes appears. An example is shown in the following figure.

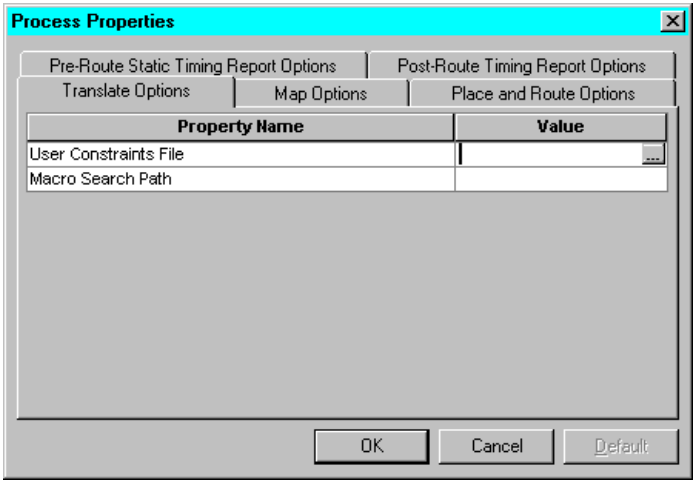


Figure 14-4 Implementation Process Properties - Standard Display

5. Click on the tab corresponding to the type of options you want to set to display the available properties. You can set properties for the following implementation option groups: Translate Options, Map Options, and Place and Route Options, Pre-Route Static Timing Report Options, and Post Route Timing Report Options. You can use the **F1** help function to display detailed descriptions of the options on each tab

   Note You can also right-click on the following processes listed under the Implement Design process to display the Process Properties dialog box for that process only: Translate, Map, Place and Route, Pre-Route Static Timing, Post Route Timing, or Multi Pass Place and Route. Multi Pass Place and Route options can only be accessed from that process.

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. If you want to access the “Advanced” options, you need to do the following before you access the Implementation Process properties dialog box.

1. Select **Edit** → **Preferences** from Project Navigator menu.
2. Click the Processes tab in the Preference dialog box.
3. Select **Advanced** for the Property Display Level. An example Advanced display is shown in the following figure.

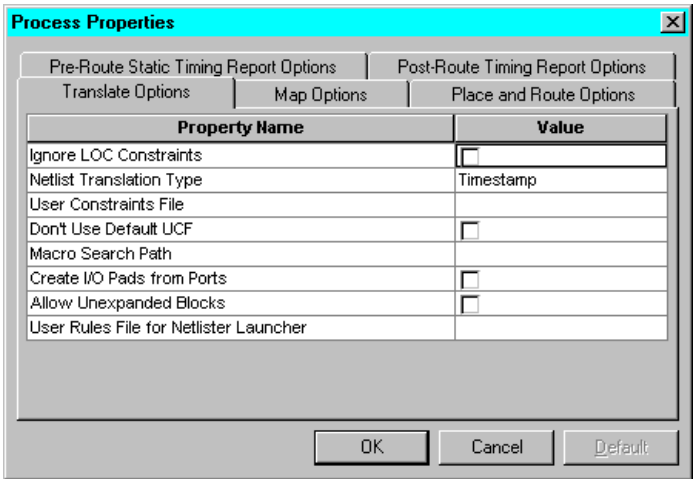


Figure 14-5 Implementation Process Properties - Advanced Display

Translate Options

Select a top-level source in the Source window, then use either of the following methods to access the list of available options for translate processing.

- Right-click on **Implement Design** in the Process window. Select **Properties** from the menu that appears. Then select the **Translate Options** tab from the Process Properties dialog box.
- Right-click on **Translate** in the Process window. Select **Properties** from the menu that appears to display a Process Properties dialog box with the Translate Options tab only.

As with all implementation properties, the translate options listed depend on whether the Standard or Advanced "property display level" is set in the Project Navigator (**Edit** → **Preferences** → **Processes**).

Standard Translate Options

An example of the standard Translate Options dialog box for Virtex is shown in the following figure. All the possible properties that can appear are described in the following sections.

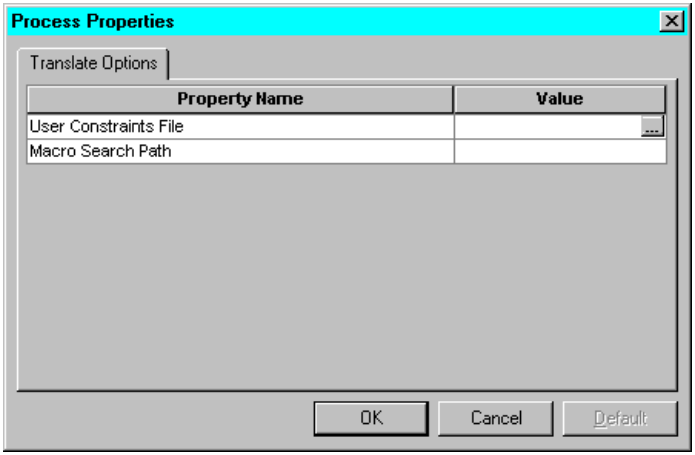


Figure 14-6 Translate Tab - Standard Display

User Constraints File

Value: Blank (default, use *project_name*.ucf) / path to a .ucf file

If the value field for this property is blank and a UCF file exists with the same base name as the project file (*project_name*, the constraints in that default UCF file (*project_name*.ucf) are automatically read. If you enter a filename with an .ucf extension here, the constraints in that file are read instead of the default UCF file. Entering a filename with an extension other than .ucf generates an error and halts processing.

Macro Search Path

Value: Blank (default) / file path

The Macro Search Path property adds the specified search path to the list of directories to search when resolving file references (that is, files specified in a schematic with a FILE=filename property). This option also supplies paths for macros (*project_name*.nmc) or other directories containing NGO files.

To specify multiple search paths, type in each directory name separated by a semicolon (;). A semicolon is automatically appended when you use the Browse button to select multiple search paths.

Advanced Translate Options

An example of the advanced Translate Options dialog box for Virtex is shown in the following figure. All the possible properties that can appear are described in the following sections.

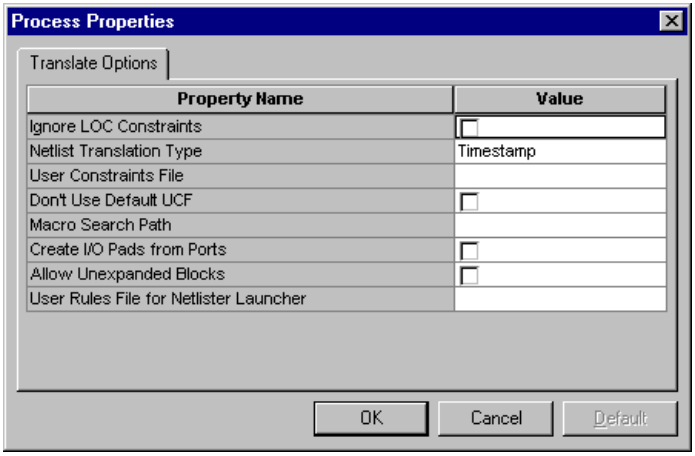


Figure 14-7 Translate Options Tab - Advanced Display

The Advanced options include all options described in the “Standard Translate Options” section plus the options described in the following sections.

Ignore LOC Constraints

Value: Disabled (default) / Enabled

If enabled (checked), this option eliminates all location constraints (LOC=) found in the input netlist or UCF file. Use this option when you migrate to a different device or architecture, because locations in one architecture may not match locations in another.

Netlist Translation Type

Value: Timestamp (default) / On / Off

The Netlist Translation Type property determines how timestamps in NGO files are treated. A timestamp is information in a file that indicates the date and time the file was created. When the “Timestamp” value is set, normal timestamp checks are done and NGO files are

updated according to their timestamps. The “On” option translates netlists regardless of timestamps (rebuilding all NGO files). The “Off” option does not rebuild an existing NGO file regardless of its timestamp.

User Constraints File

Value: Blank (default, use *project_name*.ucf) / path to a .ucf file

See the “Standard Translate Options” section for a description of this property.

Don’t Use Default UCF

Value: Disabled (default) / Enabled

The Don’t Use Default UCF property controls whether the default user constraints file (*project_name*.ucf) is used during implementation. This option is provided to allow you to disable UCF file use completely. If you select this option and do not designate a UCF file in the User Constraints File property, neither the default UCF nor any other UCF file is used for implementation processing.

Note If you specify any UCF file in the User Constraints File property and check this switch, the UCF file entered there is used.

Macro Search Path

Value: Blank (default) / file path

See the “Standard Translate Options” section for a description of this property.

Create I/O Pads from Ports

Value: Disabled (default) / Enabled

When enabled, the Create I/O Pads from Ports option adds PAD properties to all top level port signals. Select this option if your simulation netlist format is an EDIF file in which PAD symbols were translated into ports.

Allow Unexpanded Blocks

Value: Disabled (default) / Enabled

By default, the Translate process generates an error if a block in the design cannot be expanded to NGD primitives. If this error occurs, an NGD file is not written. You can enable the Allow Unexpanded Blocks option to have the Translate process generate a warning instead of an error. In this case, it writes the NGD file containing the unexpanded blocks. This option allows you to perform preliminary mapping, placement and routing, timing analysis, or simulation on a design even though the design is not complete.

User Rules for Netlister Launcher

Value: Blank (default) / Path to .urf file

The User Rules File for Netlister Launcher property allows you to control how the translate process parses files. The User Rules File specifies the acceptable netlist input files, the netlist readers that read these files, and the default netlist reader options. This file also allows you to specify third party tool commands for processing designs.

Note Note: The user rules file must have a .urf extension. If you specify a user rules file with no extension, the .urf extension is automatically added to the file name. If you specify a file name with an extension other than .urf, you get an error message and the translate process does not run.

Map Options

Select a top-level source in the Source window, then use either of the following methods to access the list of available options for map processing.

- Right-click on **`Implement Design`** in the Process window. Select **`Properties`** from the menu that appears. Then select the **`Map Options`** tab from the Process Properties dialog box.
- Right-click on **`Map`** in the Process window. Select **`Properties`** from the menu that appears to display a Process Properties dialog box with the Map Options tab only.

As with all implementation properties, the options listed depend on whether the Standard or Advanced "property display level" is set in the Project Navigator (**`Edit`** → **`Preferences`** → **`Processes`**).

Standard Map Options

An example of the standard Map Options dialog box for Virtex is shown in the following figure. All the possible properties that can appear are described in the following sections.

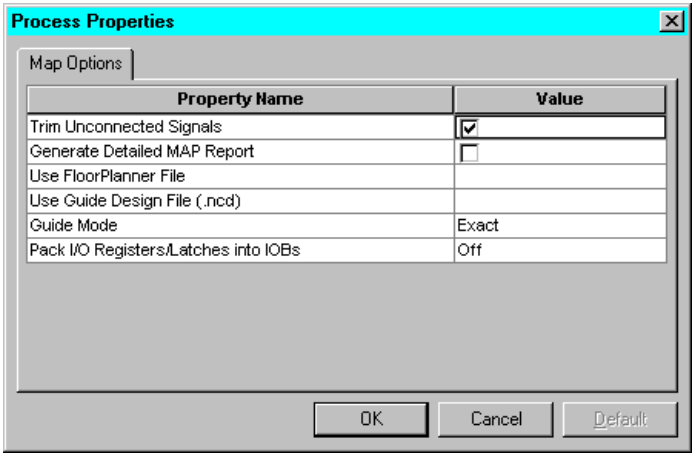


Figure 14-8 Map Options - Standard Display

Trim Unconnected Signals

Value: Enabled (default) / Disabled

When the Trim Unconnected Signals option is enabled, unconnected components and nets are trimmed from the design before mapping occurs. When disabled, unconnected components and nets. Disabling this option is useful for estimating the logic resources required for a design and for obtaining timing information on partially finished designs. When implementing an unfinished design, disable this option to prevent partial logic from being trimmed.

Generate Detailed Map Report

Value: Disabled (default) / Enabled

Enable the Generate Detailed Map Report option if you want to have the Map Report include the following information on the mapping process: signal and symbol cross-references, expanded logic, redundant blocks removed, and signals merged.

Use FloorPlanner File

Value: Blank (default) / Path to .mfp file

The User Floorplanner File property controls whether a Map Floorplanner File (MFP), which is generated by the Floorplanner, is used during the mapping process. This property requires the specification of an existing MFP file created by the Floorplanner. The MFP file is essentially used as a guide file for mapping. To create an MFP, you must first have generated an NGD file and a mapped NCD file. When you have run MAP to generate an NCD file, you can open the mapped NCD file in the Floorplanner, modify the placement of components, and then generate an MFP file. You can then use the MFP file as an input file. For more information about the Floorplanner, see the *Floorplanner Reference/User Guide*.

Note If you use the Use FloorPlanner File property, you cannot use the Use Guide Design File property.

Use Guide Design File (.ncd)

Value: Blank (default) / Path to .ncd file

The Use Guide Design File property allows you to specify an optional guide design file to be fed into the place and route process. The guide file is an NCD file that is used as a template for placing and routing the input design. This is useful if minor incremental changes have been made to create a new design. To increase productivity, you can use your last design iteration as a guide design for the next design iteration; that is, your output NCD file becomes the guide design file for your next iteration of the design. If you do not specify a guide file, the implementation process is guided by the placement and routing information in the input NCD file.

Guide Mode

Value: Exact (default) / Leverage

The Guide Mode property allows you to specify the form of guided placement and routing. Exact mode locks the placement and routing of matching logic. Neither placement nor routing can be changed to accommodate the additional logic. Leverage mode specifies that PAR should try to maintain the placement and routing of the matching logic, but can change placement or routing if it is necessary in order

to place and route to completion and achieve your timing constraints (if possible).

You specify the NCD to use as a guide file in the Use Guide Design File property. If you do not specify a guide file, PAR is guided by the placement and routing information in the input NCD file.

Pack I/O Registers/Latches into the IOBs

Value: Off (default) / For Inputs Only / For Outputs Only / For Inputs and Outputs

Normally, the mapper packs flip-flops or latches within an I/O cell only if such packing is specified by your design entry method. The Pack I/O Registers/Latches into the IOBs property allows you to control the packing of flip-flops or latches within an I/O cell as follows:

- Off (the default)

  Select Off to pack flip-flops or latches as specified by your design entry method.

- Inputs Only

  Use this value to pack flip-flops or latches into input I/O cells.

- Outputs Only

  Use this value to pack flip-flops or latches into output I/O cells.

- Inputs and Outputs

  Use this value to pack flip-flops or latches into both input and output I/O cells.

Use Generic Clock Buffers (BUFGS) in Place of BUFGPs

Value: Disabled (default) / Enabled

For Spartan, SpartanXL, and XC4000 architectures, the Use Generic Clock Buffers (BUFGs) in Place of BUFGP/BUFGS property enables the use of BUFGs instead of BUFGPs and BUFGSs.

Advanced Map Options

An example of the advanced Map Options dialog box for Virtex is shown in the following figure. All the possible properties that can appear are described in the following sections.

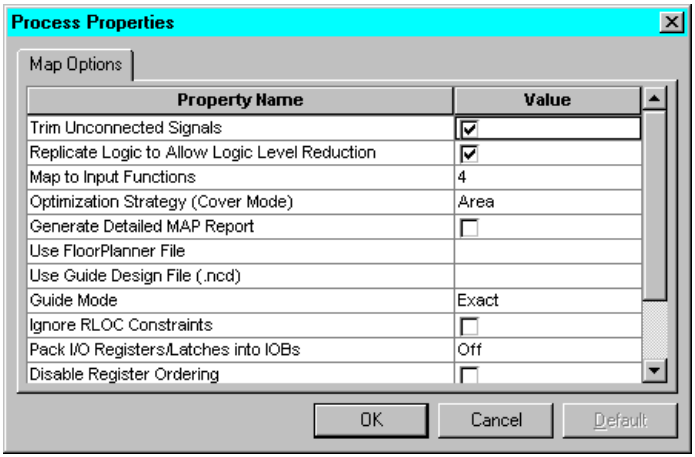


Figure 14-9 Map Options - Advanced Display

Trim Unconnected Signals

Value: Enabled (default) / Disabled

Refer to the “Standard Map Options” section for information on this option.

Replicate Logic to Allow Logic Level Reduction

Value: Enabled (default) / Disabled

When the Replicate Logic to Allow Logic Level Reduction option is enabled, single drivers that drive multiple loads are replicated and mapped as separate components that drive individual loads. This option is useful for creating a mapping strategy that may more readily meet your timing constraints. It reduces the number of logic elements through which a signal must pass, thereby eliminating path delays.

Map to Input Functions

Value: 4 (default) / 5 / 6

For Spartan2, Virtex, VirtexE, and Virtex2 architectures, you can specify the maximum size function that is covered. The default is 4. Covering to 5 or 6 input functions results in the use of F5MUX and F6MUX.

Map to 5-Input Functions

Value: Disabled (default) / Enabled

For Spartan, SpartanXL, and XC4000 architectures, you can enable the Map to 5-input Functions option to map each five-input logic function to a single CLB. This can sometimes reduce the number of cell-to-cell delays at the expense of increased CLB count.

Optimization Strategy (Cover Mode)

Value: Area (default) / Speed / Balanced / Off

The Optimization Strategy property allows you to specify criteria used during the "cover" phase of MAP. In the "cover" phase, MAP assigns the logic to CLB function generators (LUTs). The available values are as follows.

- The Area setting, the default) makes reducing the number of LUTs (and therefore the number of CLBs) the highest priority.
- The Speed setting makes reducing the number of levels of LUTS (the number of LUTs a path passes through) the highest priority. This setting makes it easiest to achieve your timing constraints after the design is placed and routed. For most designs there is a small increase in the number of LUTs (compared to the area setting), but in some cases the increase may be large.
- The Balanced setting balances two priorities: reducing the number of LUTs and reducing the number of levels of LUTs. The Balanced option produces results similar to the Speed setting but avoids the possibility of a large increase in the number of LUTs.
- The Off setting disables optimization.

Generate Detailed Map Report

Value: Disabled (default) / Enabled

Refer to the "Standard Map Options" section for a description of this property.

Use FloorPlanner File

Value: Blank (default) / Path to .mfp file

Refer to the "Standard Map Options" section for a description of this property.

Use Guide Design File (.ncd)

Value: Blank (default) / Path to .ncd file

Refer to the “Standard Map Options” section for a description of this property.

Guide Mode

Value: Exact (default) / Leverage

Refer to the “Standard Map Options” section for a description of this property.

Ignore RLOC Constraints

Value: Disabled (default) / Enabled

You can enable the Ignore RLOC Constraints property to instruct the Map process to ignore RLOC information, which contains the relative placement of one CLB to another. When enabled, this option also instructs Map to ignore any invalid RLOC information that would result in a Map error.

Pack I/O Registers/Latches into the IOBs

Value: Off (default) / For Inputs Only / For Outputs Only / For Inputs and Outputs

Refer to the “Standard Map Options” section for a description of this property.

Disable Register Ordering

Value: Disabled (default) / Enabled

The Disable Register Ordering property allows to enable or disable register ordering. When you map a design containing registers, the mapper can optimize the way the registers are grouped into CLBs. This optimized mapping is called register ordering.

CLB Pack Factor Percentage

Value: 97 (default) / Numeric value

The CLB Pack Factor Percentage property allows you to set the percent of CLBs that are packed. Enter a numeric value in this field. By default for XC4000, Spartan, and SpartanXL devices, this field is

set to 97(%). By default for Virtex, VirtexE, Virtex2, and Spartan2 devices, this field is set to 100(%).

Tri-state Buffer Transformation Mode

Value: Off (default) / On / Aggressive / Limit

For Virtex, VirtexE, Virtex2, and Spartan2 architectures, the Tri-state Buffer Transformation Mode allows you to specify the type of bus transformation that MAP should perform. When this option is set to Aggressive, Map transforms the entire bus. When set to Limit, Map transforms only the portion of the bus that exceeds the device limit.

Use Generic Clock Buffers (BUFGS) in Place of BUFGPs

Value: Disabled (default) / Enabled

For Spartan, SpartanXL, and XC4000 architectures, refer to the "Standard Map Options" section for a description of this property.

Pre-Route Static Timing Options

Select a top-level source in the Source window, then use either of the following methods to access the list of available options for pre-route static timing processing.

- Right-click on **`Implement Design`** in the Process window. Select **`Properties`** from the menu that appears. Then select the **`Pre-Route Static Timing Options`** tab from the Process Properties dialog box.
- Right-click on **`Pre-Route Static Timing`** in the Process window. Select **`Properties`** from the menu that appears to display a Process Properties dialog box with the Pre-Route Static Timing Options tab only.

As with all implementation properties, the options listed depend on whether the Standard or Advanced "property display level" is set in the Project Navigator (**`Edit`** → **`Preferences`** → **`Processes`**).

Standard Pre-Route Static Timing Options

An example of the standard Pre-Route Static Timing Options dialog box for Virtex is shown in the following figure. All the possible properties that can appear are described in the following sections.

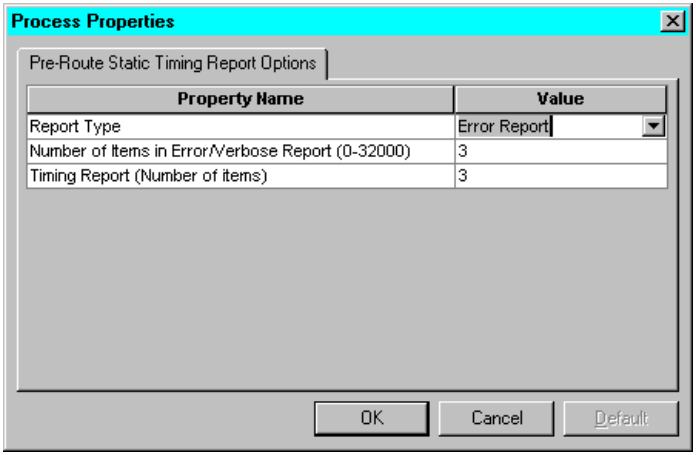


Figure 14-10 Pre-Route Static Timing Options - Standard Display

Report Type

Value: Error Report (default) / Verbose Report

The Report Type property specifies the type of report you want to run. By default, this option is set to Error Report. The Error Report lists timing errors and associated net / path delay information. Failed paths appear listed from worst-case to best-case. The Verbose Report provides more details on delays for all constrained paths and nets in the design. Entries are ordered by constraint and, within constraints, by slack.

Number of Items in Error/Verbose Report (0-32000)

Value: 3 (default) / Number from 0 through 32000

The Number of items in Error / Verbose Report property allows you to specify the number of items you want reported in those reports. The default is 3.

Timing Report (Number of Items)

Value: 3 (default) / Number from 0 through 32000

The Timing Report (Number of Items) property allows you to specify the number of items you want reported in the timing report. The default is 3.

Advanced Pre-Route Static Timing Options

An example of the advanced Pre-Route Static Timing Options dialog box for Virtex is shown in the following figure. All the possible properties that can appear are described in the following sections.

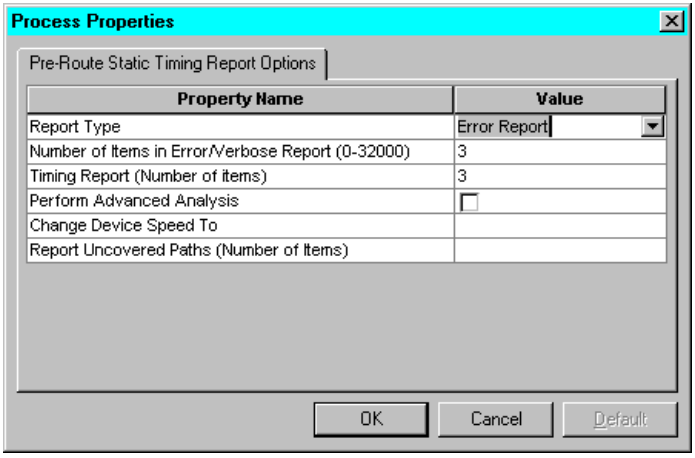


Figure 14-11 Pre-Route Static Timing Options - Advanced Display

Report Type

Value: Error Report (default) / Verbose Report

Refer to the “Standard Pre-Route Static Timing Options” section for a description of this property.

Number of Items in Error/Verbose Report (0-32000)

Value: 3 (default) / Number from 0 through 32000

Refer to the “Standard Pre-Route Static Timing Options” section for a description of this property.

Timing Report (Number of Items)

Value: 3 (default) / Number from 0 through 32000

Refer to the “Standard Pre-Route Static Timing Options” section for a description of this property.

Perform Advanced Analysis

Value: Disabled (default) / Enabled

You can enable the Perform Advanced Analysis property to generate an advanced analysis of your timing constraints. An advanced analysis enumerates all clocks and the required OFFSETs for each clock. It also contains an analysis of paths having only combinatorial logic, ordered by delay. Select this option only if you are not supplying any timing constraints in a PCF file. By default, an advanced analysis is not performed (check box is blank).

Change Device Speed to

Value: Blank (default) / Valid speed grade for the targeted device

The Change Device Speed to property allows you specify a new speed grade for your design. Changing the speed grade helps you determine if you need to target a faster device to meet your timing requirements, or if using a slower speed grade still meets timing constraints. By default, this field is blank

For more information on setting speed grades, see the TRACE chapter in the "Development System Reference Manual".

Report Uncovered Paths (Number of Items)

Value: Blank (default) / Number

The Report Uncovered Paths (Number of Items) property reports paths not affected by timing constraints. The number of paths reported depends on the value entered for this property. By default, uncovered paths are not reported.

Place and Route Options

Select a top-level source in the Source window, then use either of the following methods to access the list of available options for place and route processing.

- Right-click on **`Implement Design`** in the Process window. Select **`Properties`** from the menu that appears. Then select the **`Place and Route Options`** tab from the Process Properties dialog box.

- Right-click on **Place and Route** in the Process window. Select **Properties** from the menu that appears to display a Process Properties dialog box with the Place and Route Options tab only.

As with all implementation properties, the options listed depend on whether the Standard or Advanced "property display level" is set in the Project Navigator (**Edit** → **Preferences** → **Processes**).

Standard Place and Route Options

An example of the standard Place and Route Options dialog box for Virtex is shown in the following figure. All the possible properties that can appear are described in the following sections.

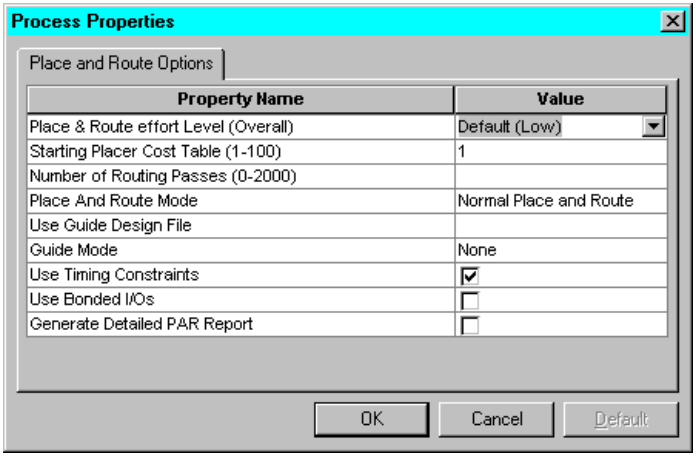


Figure 14-12 Place and Route Options - Standard Display

Place & Route Effort Level (Overall)

Value: Default (Low) (default) / Lowest / Low / Normal / High / Highest

The Place and Route Effort Level property allows you to control the placement times by selecting a less CPU-intensive algorithm for placement. You can set the placement level from Lowest (fastest run time) to Highest (best results) as defined below.

- Lowest gives the fastest runtime with lowest place and route effort. (Least complex design)

- Low (default) gives a fast runtime with some place and route optimization.
- Normal gives nominal runtime with equal place and route optimization.
- High gives a better place and route results at the expense of longer run times.
- Highest gives the best place and route results, but will incur the longest run time. (Most complex design)

If you place and route a simple design at a high level, the design is placed and routed properly, but the process takes more time than placing and routing at a lower level. If you place and route a complex design at a lower level, the design may not route to completion or may route less completely (or with worse delay characteristics) than at a higher level.

Starting Placer Cost Table (1-100)

Value: 1 (default) / Integer from 1 through 100

The Starting Placer Cost Table property allows you to specify a placement initialization value with which to begin the place and route attempts. Each subsequent attempt is assigned an incremental value based on the placement initialization value.

The number you choose corresponds to a cost table index and results in different place and route strategies. Cost tables assign weighted values to relevant factors such as constraints specified in the input file (for example, certain components must be in certain locations), the length of connections, and the available routing resources.

If cost table 100 is reached, placement does not begin at 1 again, even if command options specify that more placements should be performed. Cost tables are not an ordered set. There is no correlation between a cost table's number and its relative value.

Number of Routing Passes (0-2000)

Value: Blank (default) / Integer from 0 through 2000

The Routing Passes property allows you to set the maximum number of routing passes that the router runs in a design. The router attempts to completely route a placement with each pass during the implementation process. You can set the number of passes to a value from 0

to 2000. By default, this field is blank. You must enter a number from 0 to 2000 to incur routing passes.

If this field is left blank, the router runs until specific exit conditions are met. At Place & Route effort levels of Normal, High, or Highest, the router runs until it routes to 100% completion and meets all timing constraints or until it determines it cannot complete the routing. At the Lowest or Low levels, the router stops after a predetermined number of passes. With all settings, the router exits immediately after it routes all connections and meets all timing constraints. A higher number of passes provides better routing results at the expense of longer run times.

If you enter 0, the software continues to place and route, stopping either after the design is fully routed or after completing the iteration at cost table 100 and meeting all timing constraints.

Place and Route Mode

Value: Normal Place and Route (default) / Quick Place and Route / Quick Place and No Route / Route Only / Reentrant Route

The Place and Route Mode property allows you to specify the type of place and route you want implemented on your design. You can specify one of six options:

- Normal Place and Route (default)

  This option runs PAR with effort levels specified by the user or with the default options

- Quick Place and Route

  This option runs PAR without any timing constraints, with the lowest effort level.

- Quick Place and No Route

  This option runs the PAR process without any timing constraints, with the lowest effort level. The router will not run.

- Route Only

  This option runs PAR with effort levels you specify or with default options. The placer will not run (current placement is kept). PAR must have been run at least once to use this option.

- Reentrant Route

  This option keeps the current placement and runs the router multiple times to improve results. PAR must have been run at least once to use this option. Re-entrant (also called incremental) routing is useful if you want to manually route parts of the design and then continue automatic routing, if you halted the route prematurely (for example, with a Control-C) and want to resume, or if you want to run additional route or delay reduction passes.

Use Guide Design File

Value: Blank (default) / Path for .ncd file

The Use Guide Design File property allows you to specify an optional guide design file to be fed into the place and route process. The guide file is an NCD file that is used as a template for placing and routing the input design. This is useful if minor incremental changes have been made to create a new design. To increase productivity, you can use your last design iteration as a guide design for the next design iteration; that is, your output NCD file becomes the guide design file for your next iteration of the design.

If you do not specify a guide file, the implementation process is guided by the placement and routing information in the input NCD file.

Guide Mode

Value: None (default) / Exact / Leverage

The Guide Mode property allows you to specify the form of guided placement and routing. You can specify one of the following options:

The None (default) setting specifies that the placement and routing are not guided

Exact mode specifies that the placement and routing of the matching logic are locked. Neither placement nor routing can be changed to accommodate the additional logic

Leverage mode specifies that the implementation process tries to maintain the placement and routing of the matching logic, but changes placement or routing if it is necessary in order to place and route to completion and achieve timing objectives.

Use Timing Constraints

Value: Enabled (default) / Disabled

Select this option to produce a high-performance implementation of the design. The router uses the timing constraints in the design file to place and route the design within the specified constraints. Deselect this option to ignore timing constraints. This reduces implementation time at the expense of timing performance. By default, this option is on.

The Use Timing Constraints property allows you to specify the use of timing constraints during place and route. One method of specifying timing requirements is by entering them on the schematic. You can also specify timing requirements in constraints files (UCF and PCF). For detailed information on using Timing Constraints refer to the "Timing" chapter of the *Development System Reference Guide*.

Use Bonded I/Os

Value: Disabled (default) / Enabled

When the Use Bonded I/Os property is disabled, I/O logic that MAP has identified as internal can only be placed in unbonded I/O sites. When the option is enabled, PAR can place this internal I/O logic into bonded I/O sites in which the I/O pad is not used. The option also allows PAR to route through bonded I/O sites.

If you enable this option, you must make sure this logic is not placed in bonded sites connected to external signals, power, or ground. You can prevent this condition by placing PROHIBIT constraints on the appropriate bonded I/O sites.

Generate Detailed PAR Report

Value: Disabled (default) / Enabled

The Generate Detailed PAR Report property allows you to enable or disable the more verbose PAR Report.

Advanced Place and Route Options

An example of the advance Place and Route Options dialog box for Virtex is shown in the following figure. All the possible properties that can appear are described in the following sections.

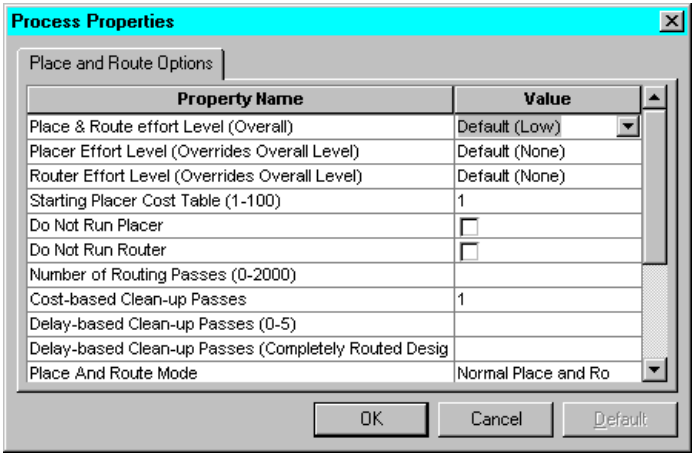


Figure 14-13 Place and Route Options - Advanced Display

Place & Route Effort Level (Overall)

Value: Default (Low) (default) / Lowest / Low / Normal / High / Highest

Refer to the “Standard Place and Route Options” section for a description of this property.

Placer Effort Level (Overrides Overall Level)

Value: Default (None) (default) / Lowest / Low / Normal / High / Highest

The Placer Effort Level property allows you to specify the placer effort separate from the router effort. The Place & Route Effort Level (Overall) property must be set to None for the Placer Effort Level Property to be in effect. If the Place & Route Effort Level (Overall) property is set to anything other than None, the value set there overrides the Placer Effort Level property.

Refer to the “Standard Place and Route Options” section for a description of the effort levels.

Router Effort Level (Overrides Overall Level)

Value: Default (None) (default) / Lowest / Low / Normal / High / Highest

The Router Effort Level property allows you to specify the router effort separate from the placer effort. The Place & Route Effort Level (Overall) property must be set to None for the Router Effort Level Property to be in effect. If the Place & Route Effort Level (Overall) property is set to anything other than None, the value set there overrides the Router Effort Level property.

Starting Placer Cost Table (1-100)

Value: 1 (default) / Integer from 1 through 100

See the "Standard Place and Route Options" section for a description of this property

Do Not Run Placer

Value: Disabled (default) / Enabled

If you enable (check) the Do Not Run Placer property, the placer does not run at all.

Do Not Run Router

Value: Disabled (default) / Enabled

If you enable (check) the Do Not Run Router property, the router does not run at all.

Number of Routing Passes (0-2000)

Value: Blank (default) / Integer from 0 through 2000

See the "Standard Place and Route Options" section for a description of this property.

Cost-Based Clean-Up Passes

Value: Device dependent (default) / Integer value from 0 through 5

The Cost-based Clean-up Passes property allows you to specify the number of time PAR should attempt to clean up any errant signals based on area availability.

If you run both cost-based cleanup passes and delay-based cleanup passes, the cost-based passes run before the delay-based passes. The valid range of cost-based cleanup passes is 0–5. The default setting is 1 for all XC4000, Spartan, and SpartanXL devices and 0 for the Spartan2, Virtex, VirtexE, and Virtex2 devices.

Following are some strategies for using the cleanup routers (either cost or delay based).

- On non-timing driven runs, cleanup routing can significantly improve delays.
- If cost-based cleanup does not yield the desired performance on a non-timing driven run, running a delay-based cleanup pass may often significantly improve circuit performance.
- For timing-driven runs, the cleanup passes can improve timing on those elements of the design that are not covered by timing constraints.
- Also, for designs in which normal iterative routing is not quite making the timing goal (but is somewhat close, say 3 - 5%) a delay-based cleanup pass can sometimes reorganize the routing enough such that follow-up re-entrant iterative routing passes are then able to meet timing.

Note This property is not recommended for use with Virtex, VirtexE, Virtex2, or Spartan2 devices. The evaluation of this option with these architectures indicates that the option creates much longer runtimes with hardly any improvement.

Delay-Based Clean-Up Passes (0-5)

Value: Blank (default) / Integer value from 0 through 5

The Delay-based Clean-up Passes property allows you to specify the number of times PAR will attempt to place any errant signals based on timing in the design.

By default, this property is blank and the router does not run any delay-based cleanup passes. If you run both delay-based cleanup passes and cost-based cleanup passes, the cost-based passes run before the delay-based passes. Typically, the first delay-based cleanup pass produces the greatest improvement, with less improvement on each successive pass. It is also possible that delay passes do not show any improvement.

If you want to run delay-based cleanup passes only on designs that have been routed to completion (100% routed), use the "Delay-based Clean-up Passes (Completely Routed Designs)" property (described below) instead of the "Delay-Based Clean-Up Passes" property.

Note The Delay-Based Clean-Up Passes property is not recommended for use with Virtex, VirtexE, Virtex2, or Spartan2. The evaluation of this option with these architectures indicates that the option creates much longer runtimes with little improvement.

Delay-Based Clean-Up Passes (Completely Routed Designs)

Value: Blank (default) / Integer value from 0 through 5

The Delay-based Clean-up Passes (Completely Routed Design) property allows you to specify the number of times PAR will attempt to place any errant signals based on timing in the design after the design has been completely routed.

The Delay-based Clean-up Passes (Completely Routed Designs) property operates in the same way as the Delay-based Clean-up Passes property described previously with the following exception. The the Delay-based Clean-up Passes property runs on *all* output designs produced by the PAR run, while the Delay-based Clean-up Passes (Completely Routed Designs) property only runs on those output designs which have been routed to completion. The number of passes is 0–5, and the default is blank.

Note This option is not recommended for use with Virtex, VirtexE, Virtex2, or Spartan2. The evaluation of this option with these architectures indicates that the option creates much longer runtimes with little improvement.

Place and Route Mode

Value: Normal Place and Route (default) / Quick Place and Route / Quick Place and No Route / Route Only / Reentrant Route

See the "Standard Place and Route Options" section for a description of this property.

Use Guide Design File

Value: Blank (default) / Path for .ncd file

See the “Standard Place and Route Options” section for a description of this property.

Guide Mode

Value: None (default) / Exact / Leverage

See the “Standard Place and Route Options” section for a description of this property.

Use Timing Constraints

Value: Enabled (default) / Disabled

See the “Standard Place and Route Options” section for a description of this property.

Use Bonded I/Os

Value: Disabled (default) / Enabled

See the “Standard Place and Route Options” section for a description of this property.

Generate Detailed PAR Report

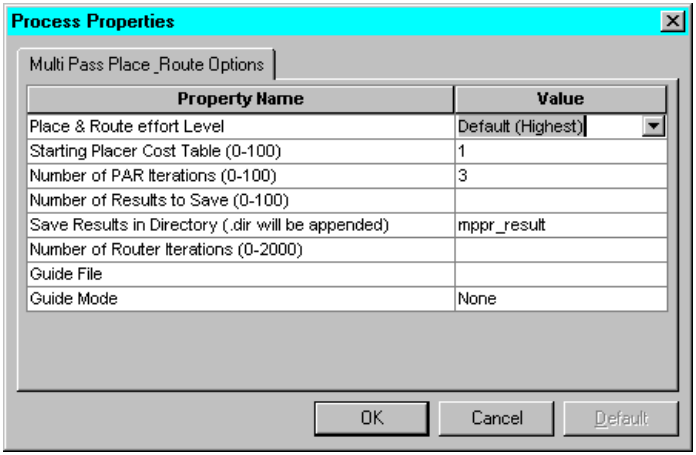
Value: Disabled (default) / Enabled

See the “Standard Place and Route Options” section for a description of this property.

Multi-Pass Place & Route Options

Use the following procedure to customize Mult-Pass Place & Route processing prior to initiating the processing.

1. Select a design source in the Source window.
2. Right-click on **`Multi-Pass Place & Route`** in the Process window.
3. Select **`Properties`** for the menu that appears.
4. The Process Properties dialog box for Multi-Pass Place and Route Options appears.



The properties you can set on the Multi-Pass Place and Route Options dialog box are described in the following sections.

Place & Route Effort Level

Value: Highest (default) / Lowest / Low / Normal / High / Highest

The Place and Route Effort Level property allows you to control the placement times by selecting a less CPU-intensive algorithm for placement. You can set the placement level from Lowest (fastest run time) to Highest (best results). Lowest gives the fastest runtime with lowest place and route effort (Least complex design). Low gives a fast runtime with some place and route optimization. Normal gives nominal runtime with equal place and route optimization. High gives a better place and route result at the expense of longer run times. Highest gives the best place and route results but will incur the longest run time.

Starting Placer Cost Table (0 - 100)

Value: 1 (default) / Number from 0 through 100

The Starting Placer Cost Table property allows you to specify a placement initialization value with which to begin the place and route attempts. Each subsequent attempt is assigned an incremental value based on the placement initialization value.

The number you choose corresponds to a cost table index and results in different place and route strategies. Cost tables assign weighted

values to relevant factors such as constraints specified in the input file (for example, certain components must be in certain locations), the length of connections, and the available routing resources.

If cost table 100 is reached, placement does not begin at 1 again, even if command options specify that more placements should be performed. Cost tables are not an ordered set. There is no correlation between a cost table's number and its relative value.

Number of PAR Iterations (0 - 100)

Value: 1 (default) / Number from 0 through 100

The Number of PAR iterations property allows you to specify the number of times PAR attempts to place and route signals on your design.

Number of Results to Save (0 - 100)

Value: 1 (default) / Number from 0 through 100

The Number of Results to Save property allows you to specify the number of reports that will be saved from the place and route attempts.

Save Results in Directory (.dir will be appended)

Value: mmpr_result (default) / directory name (to be created under the project directory)

The Save Results in Directory property allows you to specify the directory location to which the PAR reports are to be saved.

Number of Router Iterations (0 - 2000)

Value: Blank (default) / Number from 0 through 2000

The Number of Router Iterations property allows you to set the maximum number of routing passes that the router runs in a design. The router attempts to completely route a placement with each pass during the implementation process. By default, this field is blank. You must enter a number from 0 to 2000 to incur routing passes.

Guide File

Value: Blank (default) / NCD file

The Guide File property allows you to specify an optional guide design file to be fed into the place and route process. The guide file is an NCD file that is used as a template for placing and routing the input design. This is useful if minor incremental changes have been made to create a new design. To increase productivity, you can use your last design iteration as a guide design for the next design iteration; that is, your output NCD file becomes the guide design file for your next iteration of the design. If you do not specify a guide file, the implementation process is guided by the placement and routing information in the input NCD file.

Guide Mode

Value: None (default) / Exact / Leverage

The Guide Mode property allows you to specify the form of guided placement and routing. By default, this field is set to None; placement and routing are not guided. If set to Exact, the placement and routing of the matching logic are locked. Neither placement nor routing can be changed to accommodate the additional logic. If set to Leverage, the implementation process tries to maintain the placement and routing of the matching logic, but changes placement or routing if it is necessary in order to place and route to completion and achieve your timing constraints (if possible).

Post Route Timing Options

Select a top-level source in the Source window, then use either of the following methods to access the list of available options for pre-route static timing processing.

- Right-click on **`Implement Design`** in the Process window. Select **`Properties`** from the menu that appears. Then select the **`Post Route Timing Options`** tab from the Process Properties dialog box.
- Right-click on **`Post Route Timing`** in the Process window. Select **`Properties`** from the menu that appears to display a Process Properties dialog box with the Post Route Options tab only.

Select the Post Route Timing Options tab on the Process Properties window to access the list of available options for mapping a design. As with all implementation properties, the options listed depend on whether the Standard or Advanced "property display level" is set in the Project Navigator (**`Edit`** → **`Preferences`** → **`Processes`**).

Standard Post Route Timing Options

An example of the standard Post Route Timing Options dialog box for Virtex is shown in the following figure. All the possible properties that can appear are described in the following sections.

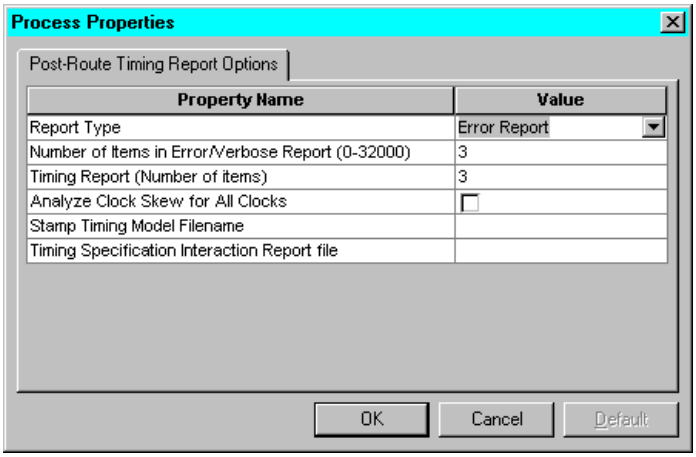


Figure 14-14 Post Route Timing Options - Standard Display

Report Type

Value: Error Report (default) / Verbose Report

The Report Type property specifies the type of report you want to run. By default, this option is set to Error Report. The Error Report lists timing errors and associated net / path delay information. Failed paths appear listed from worst-case to best-case. The Verbose Report provides more details on delays for all constrained paths and nets in the design. Entries are ordered by constraint and, within constraints, by slack.

Number of Items in Error/Verbose Report (0-32000)

Value: 3 (default) / Number from 0 through 32000

The Number of items in Error / Verbose Report property allows you to specify the number of items you want reported in those reports. The default is 3.

Timing Report (Number of Items)

Value: 3 (default) / Number from 0 through 32000

The Timing Report (Number of Items) property allows you to specify the number of items you want reported in the timing report. The default is 3.

Analyze Clock Skew for All Clocks

Value: Disabled (default) / Enabled

You can enable the Analyze Clock Skew for All Clocks property to generate a clock skew report in which the difference between the time a clock signal arrives at a source flip-flop in a path and the time it arrives at the destination flip-flop on the same clock net are analyzed.

Stamp Timing Model Filename

Value: Blank (default) / Stamp Filename

The Stamp Timing Model Filename property allows you to specify the filename of the stamp file you want to use during post-route timing. You can enter the name and location of the stamp file or click in the Value field to access a browse button to select the stamp file you want to use.

The Xilinx TRACE program generates a pair of STAMP timing model files stampfile.mod and stampfile.data that characterize the design's timing. The STAMP compiler can be used for any board when performing static timing analysis.

Timing Specification Interaction Report File

Value: Blank (default) / Stamp Filename

The Timing Specification Interaction Report File allows you to specify where the Timing Specification Interaction Report File will be placed when the post-route timing report is created. You can enter the name

and location of the file or click in the Value field to access a browse button to select the stamp file you want to use.

Advanced Post Route Timing Options

An example of the advanced Post Route Static Timing Options dialog box for Virtex is shown in the following figure. All the possible properties that can appear are described in the following sections.

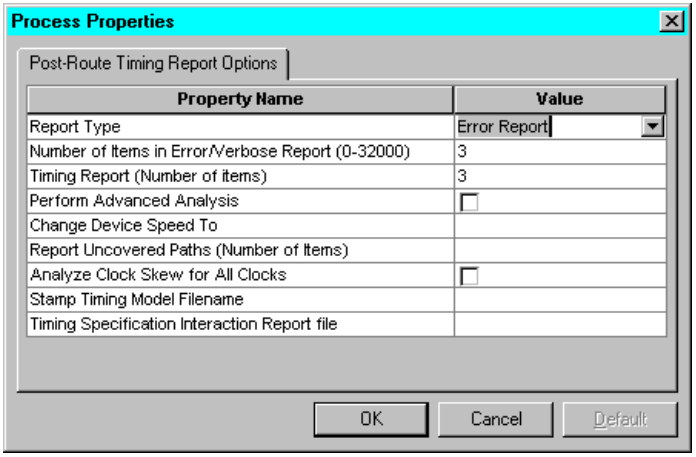


Figure 14-15 Pre-Route Static Timing Options - Advanced Display

Report Type

Value: Error Report (default) / Verbose Report

Refer to the “Standard Post Route Timing Options” section for a description of this property.

Number of Items in Error/Verbose Report (0-32000)

Value: 3 (default) / Number from 0 through 32000

Refer to the “Standard Post Route Timing Options” section for a description of this property.

Timing Report (Number of Items)

Value: 3 (default) / Number from 0 through 32000

Refer to the “Standard Post Route Timing Options” section for a description of this property.

Perform Advanced Analysis

Value: Disabled (default) / Enabled

You can enable the Perform Advanced Analysis property to generate an advanced analysis of your timing constraints. An advanced analysis enumerates all clocks and the required OFFSETs for each clock. It also contains an analysis of paths having only combinatorial logic, ordered by delay. Select this option only if you are not supplying any timing constraints in a PCF file. By default, an advanced analysis is not performed (check box is blank).

Change Device Speed to

Value: Blank (default) / Valid speed grade for the targeted device

The Change Device Speed to property allows you specify a new speed grade for your design. Changing the speed grade helps you determine if you need to target a faster device to meet your timing requirements, or if using a slower speed grade still meets timing constraints. By default, this field is blank

For more information on setting speed grades, see the TRACE chapter in the “Development System Reference Manual”.

Report Uncovered Paths (Number of Items)

Value: Blank (default) / Number

The Report Uncovered Paths (Number of Items) property reports paths not affected by timing constraints. The number of paths reported depends on the value entered for this property. By default, uncovered paths are not reported.

Analyze Clock Skew for All Clocks

Value: Disabled (default) / Enabled

Refer to the “Standard Post Route Timing Options” section for a description of this property.

Stamp Timing Model Filename

Value: Blank (default) / Stamp Filename

Refer to the “Standard Post Route Timing Options” section for a description of this property.

Timing Specification Interaction Report File

Value: Blank (default) / Stamp Filename

Refer to the “Standard Post Route Timing Options” section for a description of this property.

FPGA 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 hand-crafting for their designs. The user must understand both the details of the device architectures and how the tool can be used to refine a design.

Floorplanner (FPGAs)

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 to the target device. 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. The Floorplanner window displays the floorplan of the target device into which you place logic from the hierarchy.

You can invoke the Floorplanner by selecting a design source in the Source window and then double-clicking **`FloorPlanner`** in the Process window. The Floorplanner opens and automatically loads the project’s NGD file. An example of the Floorplanner window is shown in Figure 14-16.

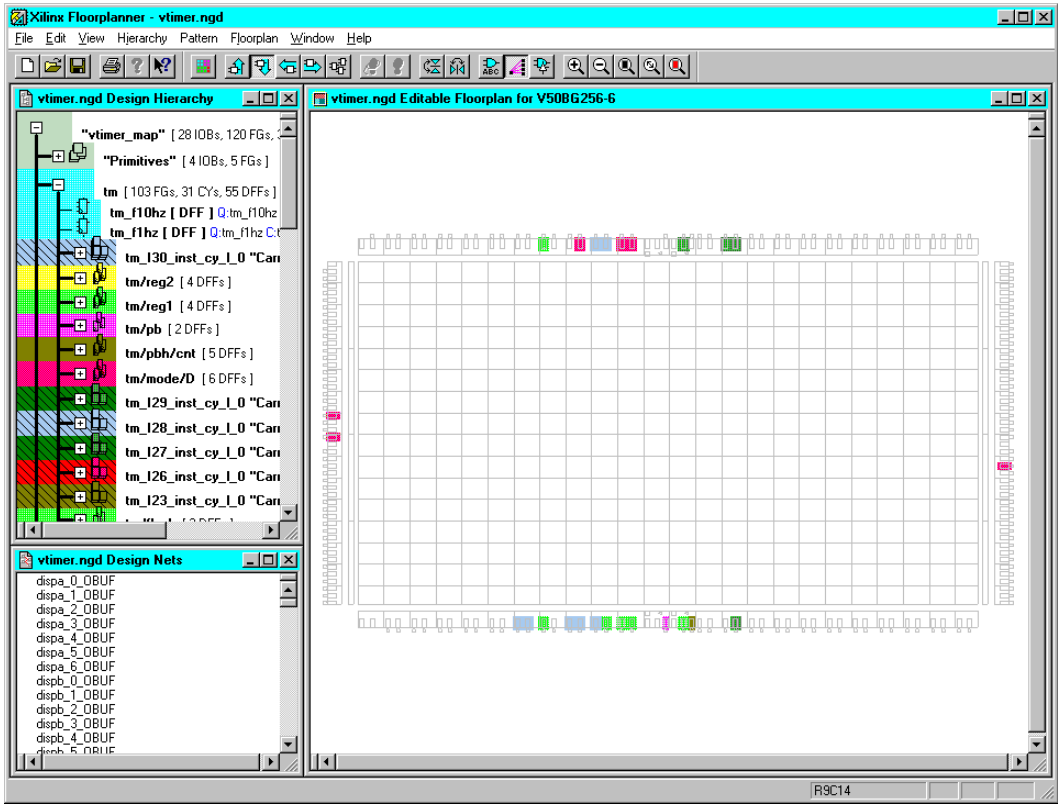


Figure 14-16 Floorplanner

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.

With the Floorplanner, you can floorplan your design prior to 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 tool 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, refer to the Floorplanner's online help or to the *Floorplanner Guide*, an online manual.

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. You can 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).

You can invoke the FPGA Editor by selecting a design source in the Source window and then double-clicking **FPGA Editor** in the Process window. The FPGA Editor opens and automatically loads the project's NCD file. An example of the FPGA Editor is shown in the following figure.

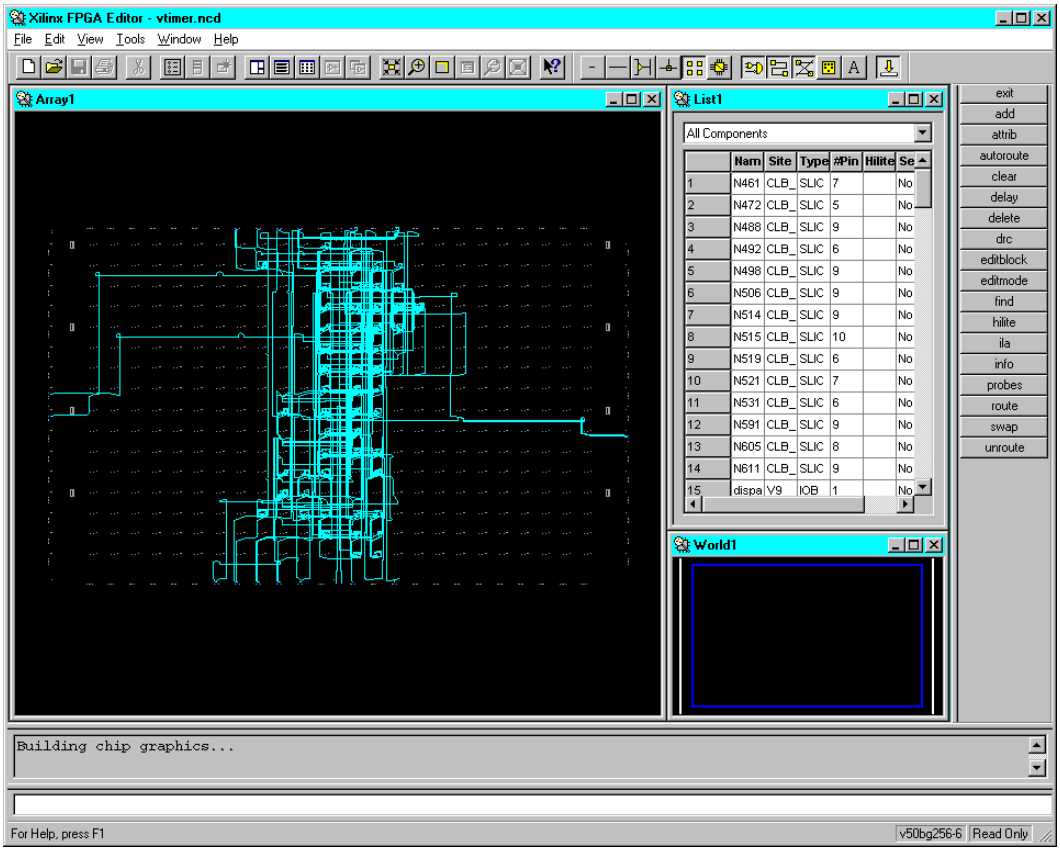


Figure 14-17 FPGA Editor

For detailed information on using the FPGA Editor, see the FPGA Editor's online help or the *FPGA Editor Guide*, an online book.

Timing Analyzer

You can use the Timing Analyzer program to perform static timing analysis on an FPGA or CPLD design. 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.

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.

You can invoke the Timing Analyzer by selecting a design source in the Source window and then double-clicking **Timing Analyzer** in the Process 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's online help or the *Timing Analyzer Guide*, an online manual.

CPLD Implementation Flow

The following figures shows the implementation portion of the Process window for a design that targets a CPLD device. The implementation processes are under “Implement Design.”

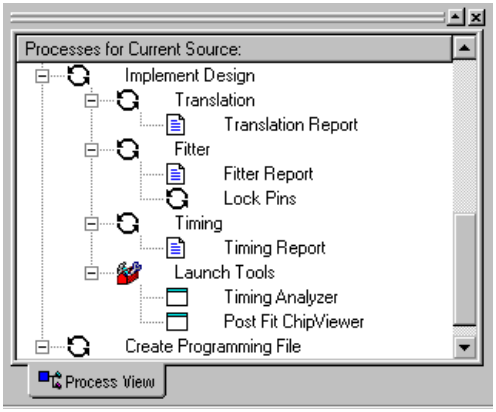


Figure 14-18 CPLD Implement Design Processes

When you select a source and then 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 “CPLD Implementation Options” section.

Xilinx implementation tools are used to process your design. If you are familiar with the Xilinx Alliance product, you will notice that the command line entries that appear in the Transcript window as each function is run correspond to the Alliance development system commands.

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.

Translation

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 types of files 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 schematic editor tool 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 file containing the implementation of a module in the design. LogiBLOX creates an NGC file to define each module.

The following types of files 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. Refer to the "Viewing Implementation Reports" section for information on this report.

For a complete description of Translate process, refer to the "NGDBuild" chapter of the *Development System Reference Guide.*

Fitter

The NGD file output by the Translate process is input to the Fit process. During Fit, the CPLD Fitter minimizes and collapses the combinational 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 implementation Process Properties dialog box.

The following file types are *output files* for the Fit process:

- Fitter Report (RPT file)

  The Fitter Report (*top_source_name*.rpt) shows you information such as the type and quantity of device resources used and the resulting pinout. Refer to the "Viewing Implementation Reports" section for information on this report.

- Guide file (GYD file)

  The Guide file (*top_source_name*.gyd) is used to lock signal names to device pins, allowing you to keep the device pinouts during subsequent design iterations. The Guide file contains all resulting pinout information required to reproduce the current pinout if the "Lock Pins" option is specified during the next invocation of the implementation processes for the same design name. The Guide file 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, refer to *CPLD Design Techniques* and *CPLD Flow Tutorial* in the on-line help (**Help** → **Foundation ISE Help Contents** from the Project Navigator).

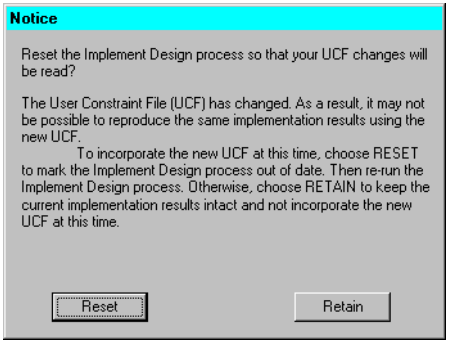
Lock Pins (Optional)

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 read from a fitted GYD file (*top_source_name*.gyd).

When you are ready to commit the pinout of a design, select the top-level source file in the Source window and then double-click the **Backannotate Pin Locs** process. 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 implementation Process Properties). This pinout will then be applied to all subsequent design implementations that you run.

For CPLDs, you can use an external guide file to lock pins, if desired. To specify the location of the external guide file, right-click on the **Lock Pins** process and select **Properties**. Enter the file path in the **Use External GYD File** field in the Process Properties dialog box that appears and then click **OK**. Double-click on **Lock Pins** to run that process with the specified GYD file.

The Notice informational window shown in the following 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 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

The Backannotate Pin Locs Report is displayed automatically upon completion of the Lock Pins process. The Backannotate Pin Locs Report (*top_source_name*.lck for FPGAs, t*top_source_name*._lc for CPLDs) is displayed in the ISE Report Viewer.

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

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. Prior to 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” and 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.

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.

CPLD Implementation Reports

This section contains descriptions of the reports available for information on the implementation processing of your CPLD design.

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

Fitting Report (CPLDs)

The Fitting Report (*top_source_name*.rpt) lists summary and detailed information about the logic and I/O pin resources used by the design, including the pinout, error and warning messages, and Boolean equations representing the implemented logic.

Timing Report

The Timing Report (*top_source_name*.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.

CPLD Implementation Options

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.

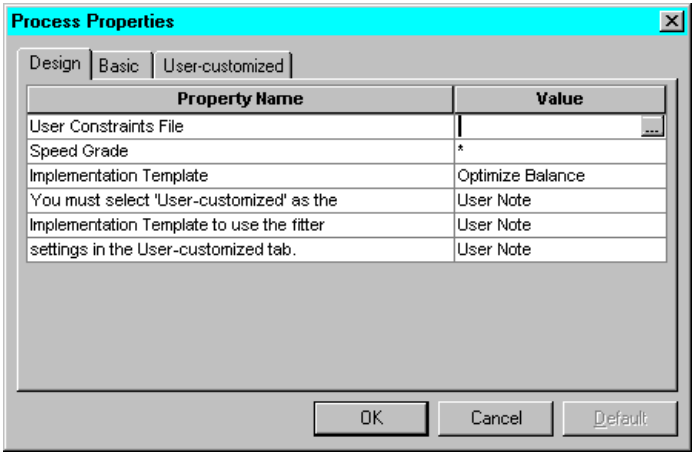
Accessing the Implementation Process Properties Dialog Box.

CPLD implementation properties are set as follows.

1. Click on a design source file in the Source 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 on **`Implement Design`** in the Process window.
3. Click on **`Properties`** from the pull-down menu that appears for Implement Design.
4. The Process Properties dialog box for Implement Design appears as shown in the following figure.



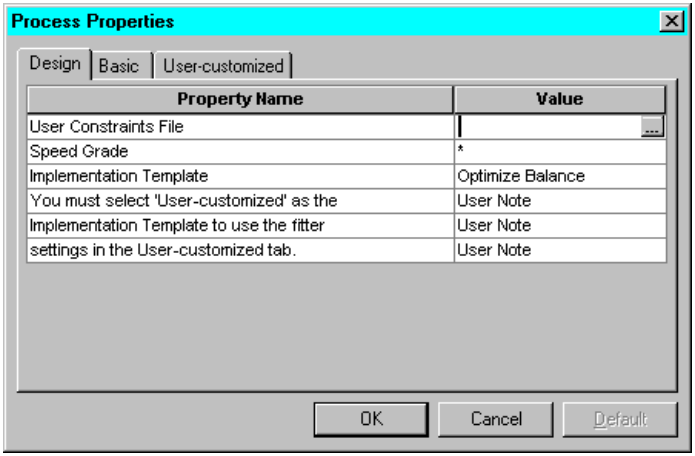
5. Click on a tab to access the list of properties you can set for the following implementation options: Design, Basic, User-Customized.

Standard and Advanced Properties

The options listed in the CPLD implementation 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).

Design Properties

From the Design tab on the CPLD implementation Process Properties dialog box, you can specify the User Constraint File name, select one of the available speed grades for the targeted device, and choose between a preset template or user-customized settings for the fitter optimization options.



User Constraints File (UCF File)

Value: Blank (default) / User Constraints File (.ucf file)

The User Constraints File (default name is *top_source_name*.ucf) is an ASCII file that you create with a text editor or using the Xilinx Constraints Editor. The file contains timing and other constraints that affect how specific design elements are implemented in the target device. The constraints in the UCF file are added to the information in the output NGD file.

Speed Grade

Value: Valid speed grade for targeted device

The Speed Grade property allows you to change the speed grade used during timing analysis. Changing the speed grade helps you determine if you need to target a faster device to meet your timing requirements, or if using a slower speed grade still meets timing constraints. You must specify a valid speed grade for the device you are targeting. Click the pull-down menu in the Value field for a list of valid speed grades for the targeted device. If you are currently targeting a device that is to be selected automatically by the Fitter ("AUTO" in the targeted device name), an asterisk (*) appears in the Value field and the Value field cannot be modified. In this case, the fastest speed grade available for the device selected by the Fitter is automatically used.

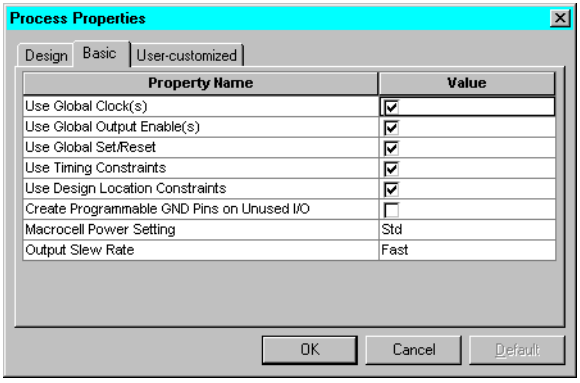
Implementation Template

Value: Optimize Balance (default) / Optimize Speed / Optimize Density / User Customized

Use the Implementation Template property to set the optimization style to be used to fit your design. You can optimize the design for speed or density. The default is to balance the optimization between speed and density. If you want to use the fitter settings specified in the User-customized tab, you must select User Customized for the Implementation Template.

Basic Properties

From the Basic tab on the CPLD implementation Process Properties dialog box, you can specify various Fitter control options that are independent of the selected optimization template.



Use Global Clock(s)

Value: Enabled (default) / Disabled

The Use Global Clock(s) property controls the option to allow the fitter to assign input pins used as clocks to dedicated global clock (GCK) pins of the device. If this option is disabled, only pins identified with BUFG=CLK property in the design (or UCF file) will be assigned to GCK device pins.

Use Global Output Enable(s)

Value: Enabled (default) / Disabled

The Use Global Output Enable(s) property controls the option to allow the fitter to assign input pins used as output enable control to dedicated global OE (GTS) pins of the device. If this option is disabled, only pins identified with the BUFG=OE property in the design (or UCF file) will be assigned to the GTS device pins.

Use Global Set/Reset

Value: Enabled (default) / Disabled

The Use Global Set/Reset property controls the option to allow the fitter to assign an input pin used as register asynchronous reset or preset control to the dedicated global set/reset (GSR) pin of the device. If this option is disabled, only a pin identified with the BUFG=SR property in the design (or UCF file) will be assigned to the GSR device pin.

Use Timing Constraints

Value: Enabled (default) / Disabled

The Use Timing Constraints property controls the option to indicate that you want the software to use your timing constraints file to perform timing-driven optimization in the fitting of your design. Disable this option to temporarily ignore all timing constraints in your design or your user constraints file (UCF).

Use Design Location Constraints

Value: Enabled (default) / Disabled

The Use Design Location Constraints property controls the option to indicate that you want the fitter to use your pinout or macrocell location information in the design file or constraint file. Disable this option to temporarily ignore all location constraints, allowing the fitter to place pins and logic anywhere.

Create Programmable GND Pins on Unused I/O

Value: Disabled (default) / Enabled

The Create Programmable GND Pins on Unused I/O property controls the option to indicate that you want all unused I/O pads to be configured as ground pins. This can reduce ground bounce.

Macrocell Power Setting

Value: Std (default) / Low / Timing Driven

The Macrocell Power Setting property controls the device power consumption. By default, this option is set to Std (standard). Std sets the power mode for the macrocells to higher speed and higher power. Low reduces power consumption and speed.Timing Driven sets the power consumption based on your timing constraints.

Note Any explicit power control statements in the design or constraints file have precedence over the Macrocell Power Setting.

Output Slew Rate

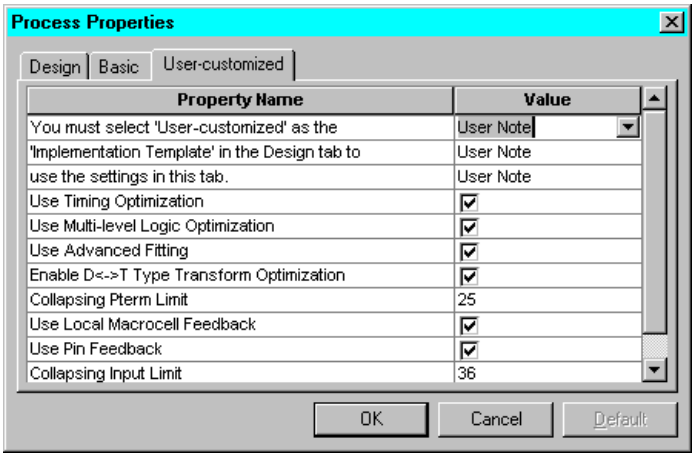
Value: Fast (default) / Slow / Timing Driven

The Output Slew Rate property controls the output slew rate. You can control the transition time of device output pins by setting the slew rate to Slow or Fast. Fast uses more current and enables faster logic responses. Slow limits the slew rate to reduce output switching surges in the device. Timing Driven automatically reduces slew rate on output pins covered by timing specifications that can meet speed requirements while operating with slow slew rates.

Note Any explicit slew rate control statements in the design or constraints file have precedence over the Output Slew Rate.

User-Customized Properties

You must select “User-Customized” as the Implementation Template on the Design tab if you want to use the fitter settings specified in the “User-customized” tab.



Use Timing Optimization

Value: Enabled (default) / Disabled

The Use Timing Optimization property allows you to enable the global timing optimization performed by the fitter. If this option is disabled, only paths with T-specs specified in the design are optimized to improve timing. By default, this option is enabled and the design is optimized using the speed template. Turn this option off to optimize using the density template.

Use Multi-level Logic Optimization

Value: Enabled (default) / Disabled

The Use Multi-level Logic Optimization property allows you to specify whether to enable or disable multi-level logic optimization.

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

Multi-level Logic Optimization optimizes combinatorial logic from your design. Combinatorial logic includes the following types of logic.

- Register-to-register logic
- Pad-to-register logic
- Register-to-pad logic
- Pad-to-pad logic

Multi-level Logic Optimization operates on combinatorial logic according to the following rules.

- If timing constraints are set, the program optimizes for speed to meet timing constraints.
- If timing constraints are not set, the program optimizes either for speed or density, depending on the user setting for the Use Timing Optimization option.
  - If Use Timing Optimization is turned on, the combinatorial logic will be mapped for speed.
  - If Use Timing Optimization is turned off, the combinatorial logic will be mapped for density. The goal of optimization will then be to reduce the total number of p-terms.
- Logic marked with the attribute MINIMIZE=OFF will not be extracted or optimized.

Use Advanced Fitting

Value: Enabled (disabled) / Disabled

The Use Advanced Fitting property allows you to specify whether to enable or disable advanced fitting. Select this option to enable an advanced fitting strategy that favors placing signals with common inputs in the same function block. This usually allows you to pack more logic into the same device. Disable this option if the software has trouble fitting a design that used to fit with an older version of software.

Enable D <--> T Type Transform Optimization

Value: Enable (default) / Disable

The Enable D<->T Type Transform Optimization property allows you to specify whether to enable or disable this optimization. If D to T type transform is enabled (checked), the fitter transforms between D-type and T-type registers.

Collapsing Pterm Limit

Value: 25 (default) / Integer from 2 through 90

The Collapsing Pterm Limit property allows you to specify the product term (pterm) limit.

When a larger combinatorial logic function consisting of several levels of AND-OR logic is completely collapsed (flattened), the number of product terms required to implement the function may grow considerably. By default, the fitter limits the number of p-terms used as a result of collapsing to 25. If the collapsing of a logic level results in a logic function consisting of more than the p-term limit (after Boolean reduction), then the collapsing of that logic level is not performed and the function will be implemented using two or more levels of AND-OR logic.

Use Local Macrocell Feedback

Value: Enabled (default) / Disabled

For XC9500 devices (except XC9536), the Use Local Macrocell Feedback property allows you to enable the software to use local feedback whenever possible. The local feedback path takes less time than the global feedback path. Using local feedback can speed up your design but can make it difficult to keep the same timing after a design change.

Use Pin Feedback

Value: Enabled (default) / Disabled

For XC9500 devices, the Use Pin Feedback property allows you to enable the software to use local I/O pin feedback whenever possible. The software uses the pin feedback path instead of the FastCONNECT path for output pin signals that do not have 3-state control or slow slew rate (by default).

Collapsing Input Limit

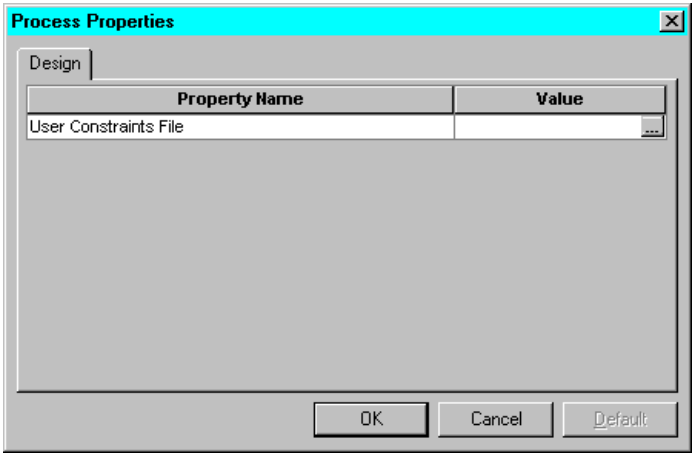
Value: Device dependent number (default) / device dependent number (Integer from 2 through 36 or from 2 through 54)

The Collapsing Input Limit property allows you to specify the maximum number of function block inputs allowed as a result of logic collapsing.

This option controls the degree to which a design netlist is flattened. A logic gate can collapse forward into a subsequent gate only if the number of inputs in the resulting logic function does not exceed the input limit. If the path delay of a logic function is not acceptable, increase the input limit to allow the larger functions to be further flattened. For XC9500 devices, choose a number from 2 to 36. The default is 36. For XC9500XL and XC9500XV devices, choose a number from 2 to 54. The default is 54.

Translation Options

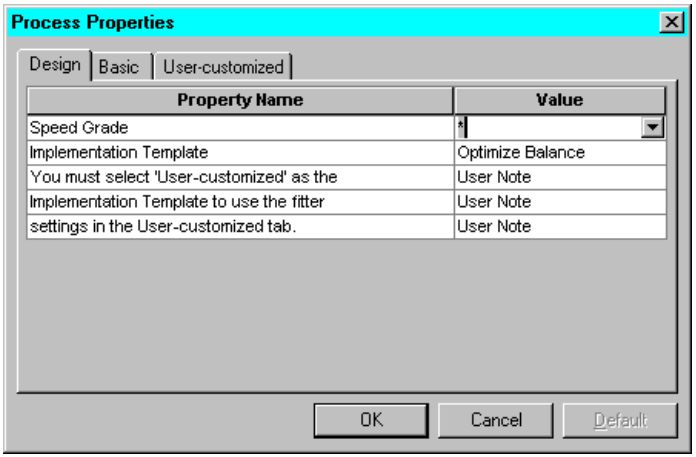
To access a Process Properties dialog box with options specific to the Translation process, select a top-level source in the Source window, then right-click on **`Translation`** in the Process window. Select **`Properties`** from the pull-down menu that appears to display the Design tab with only translation-related properties as shown in the following figure.



Refer to the “Design Properties” section for information on the User Constraints File option.

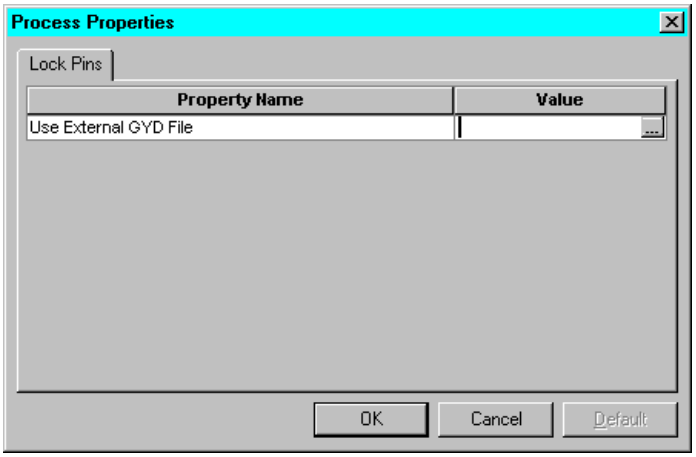
Fitter Options

To access a Process Properties dialog box with options specific to the Fitter process, select a top-level source in the Source window, then right-click on **Fitter** in the Process window. Select **Properties** from the pull-down menu that appears to display the Implement Design Properties dialog box. Note that the Design tab has only fitter-related properties as shown in the following figure. The other tabs are the same as described previously. Refer to the “Design Properties”, “Basic Properties”, and “User-Customized Properties” sections for information on the options on those tabs.



Lock Pins Options

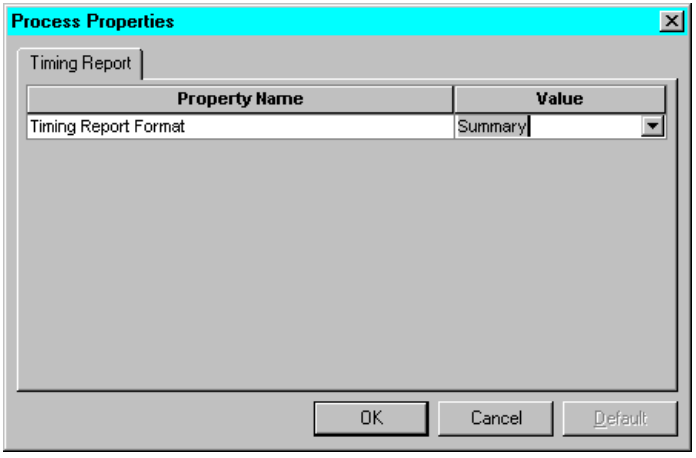
To access the Lock Pins Process Properties dialog box, select a top-level source in the Source window, then right-click on **Lock Pins** in the Process window. Select **Properties** from the pull-down menu that appears to display the Lock Pins Process Properties dialog box shown in the following figure.



In the Value field, enter the file path of the external guide file to use with the Lock Pins process. Refer to the "Lock Pins (Optional)" section for information on the Lock Pins process.

Timing Options

To access the Timing Process Properties dialog box, select a top-level source in the Source window, then right-click on **Timing** in the Process window. Select **Properties** from the pull-down menu that appears to display the Timing Report Process Properties dialog box shown in the following figure.



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.

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 hand-crafting for their designs. The user must understand both the details of the device architectures and how the tool can be used to refine a design.

Timing Analyzer

You can use the Timing Analyzer program to perform static timing analysis on an FPGA or CPLD design. 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.

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.

You can invoke the Timing Analyzer by selecting a design source in the Source window and then double-clicking **`Timing Analyzer`** in

the Process 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's online help or the *Timing Analyzer Guide*, an online manual.

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. You can examine both pre-fitting and post-fitting results.

More information on using the CPLD ChipViewer is available in that tool's online help (**Tools** → **Implementation** → **CPLD Chip-Viewer** → **Help**) or from the Foundation Series ISE Help menu accessed by **Help** → **Foundation ISE Help Contents** → **Advanced Tools** → **ChipViewer**.

Chapter 15

Snapshots

A snapshot is a picture of the design at a particular stage. It contains everything that was in the project directory and a copy of all remote sources used in the design. The project properties are also captured. A snapshot allows you to preserve a particular state of the design for later use.

This chapter contains the following sections.

- "Archives vs. Snapshots"
- "Taking a Snapshot"
- "Renaming a Snapshot"
- "Deleting a Snapshot"
- "Viewing Snapshot Contents"
- "Replacing the Current Project with a Snapshot"

Archives vs. Snapshots

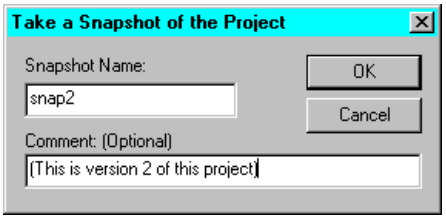
To save a specific revision of your project, you can archive it or take a snapshot of it. When you archive a project (**`Project`** → **`Archive`** from the Project Navigator menu), you create a .zip file for that version and place it in a specified directory. To use an archive or see the files in it, you must unzip it.

When you take a snapshot of the project, however, the snapshot becomes part of the project. It is accessible from the Snapshot View of the Source window. You can open it at anytime 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

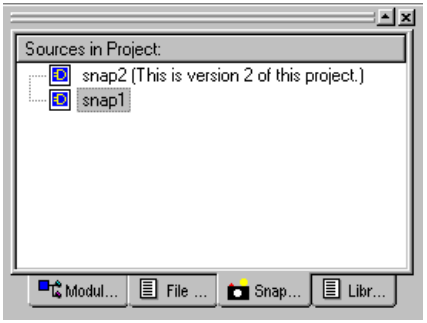
Use the following procedure to take a snapshot of the current version of your project.

1. Select **Project** → **Take Snapshot** from the Project Navigator menu.
2. When the Take a Snapshot of the Project dialog box appears, enter a name and any comments to describe the current version. By default, snapshots are named “snap1,” “snap2,” and so on. The name appears in the Snapshot View of the Source window. Comments appear after the name in the Source window if the Files Names command in the View menu is enabled.



3. Click **OK**.

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 project’s snapshots are listed on the Snapshot View tab in the Source window as shown in the following figure.



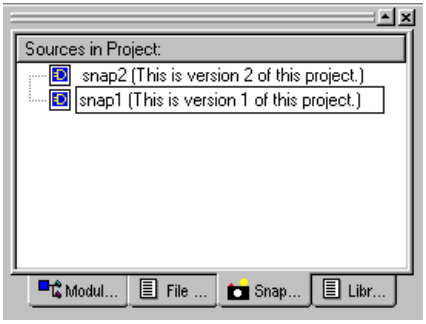
All source files that have been added to the project (including user documents) are copied for the snapshot. The process files necessary to recreate the state of the project when you made the snapshot are also copied.

Renaming a Snapshot

You can rename a snapshot or add a comment for it using the following procedure.

1. Click the Snapshot View tab in the Source window.
2. Click on a snapshot name to select it.
3. Select **Source** → **Rename** from the Project Navigator menu.
4. Modify the name or add a comment as desired and then press **Enter**.

   Note Snapshot names cannot contain spaces. You must keep the parentheses around the comment.



Deleting a Snapshot

You can delete a snapshot using the following procedure.

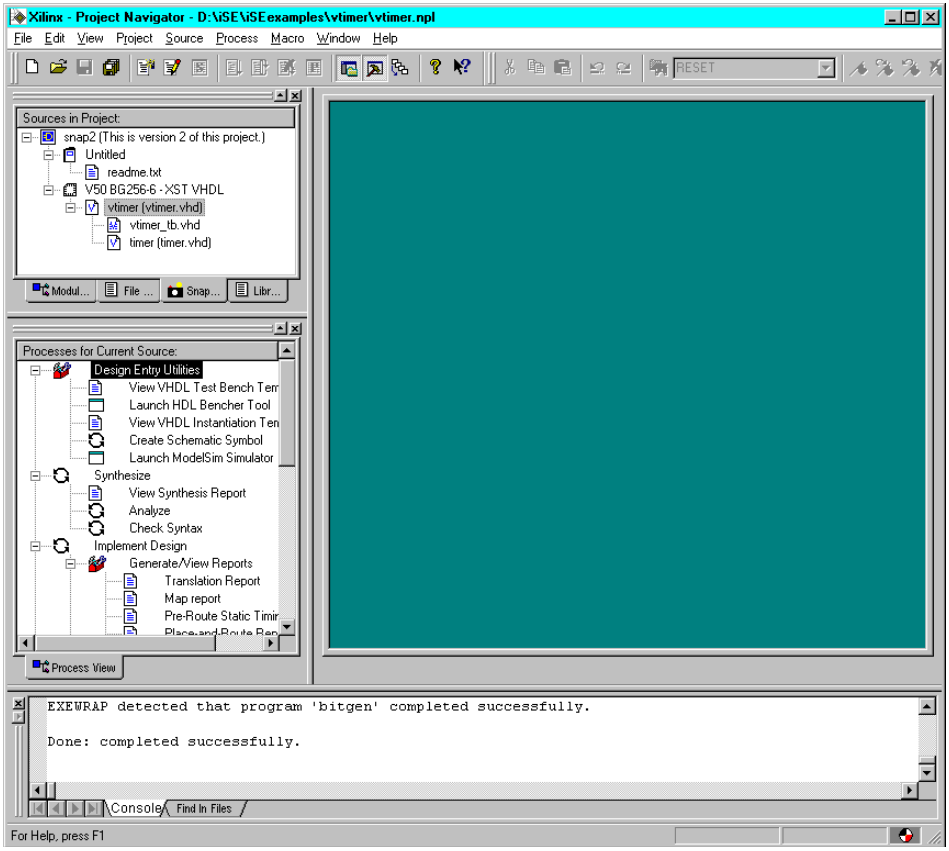
1. Click the Snapshot View tab in the Source window.
2. Click on a snapshot name to select it.
3. Select **Source** → **Remove** from the Project Navigator menu.
4. Confirm the deletion of all the files.

The snapshot and all its associated files are deleted.

Viewing Snapshot Contents

You can open and view the files contained in a snapshot using the following procedure.

1. Select the Snapshot View tab in the Source window. The Snapshot View tab displays the snapshot names.
2. To select a snapshot, click on its name.
3. Select **`Source`** → **`Open`** from the Project Navigator.
4. The sources for the selected snapshot appear in a hierarchical display under the snapshot name in the Source window. Highlight a source to display the processes for that source (with check marks status) in the Process window.



The snapshot is view-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 “Replacing the Current Project with a Snapshot” section.

Viewing Source File Contents

When the hierarchical contents of a snapshot are displayed in the Source window, simply double-click on 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 view-only files.

Viewing Report Contents

When the hierarchical contents of a snapshot are displayed in the Source window, simply click on a source to select it. If any reports have been run on that source (as indicated in the Process window), double-click on the desired report to open it in the Report Browser.

Because the Report Browser can have multiple reports open at once, you can open reports from multiple snapshots for comparison.

Replacing the Current Project with a Snapshot

If you want to make changes to the project version represented by in a specific snapshot, you can make that snapshot version the current version of the project.

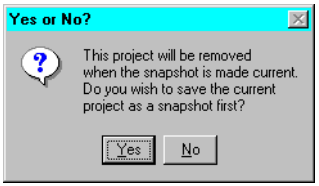

Warning

This has the result of replacing 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.

Use the following procedure to replace the current version of a project with a previous snapshot.

1. Click the Snapshot View tab in the Source window.
2. Click on a snapshot name to select it.
3. Select **`Project`** → **`Replace with Snapshot`** from the Project Navigator menu.

4. In the Yes or No dialog box that appears, select **Yes** to save your current project as a snapshot or **No** to replace your current project without saving it as a snapshot.



   If you select **Yes**, the "Taking a Snapshot of the Project" dialog box appears 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.

Chapter 16

Programming the Device

When the design meets your requirements, the last step in its processing is programming the target device.

- “Creating FPGA Programming Files”
- “Creating CPLD Programming Files”
- “Programming Tools”

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 for your design, use the following procedure.

1. Select the top-level source for the project in the Source window.
2. Click **`Create Programming File`** in the Process window.
3. Click **`Process`** → **`Run`** in the Project Navigator menu. (An alternative method is to double-click on **`Creating Programming File`** in the Process window.)
4. The programming file creation process runs. If there are no errors, the *top_source_name*.bit file is created.

5. To view the Programming File Report in the ISE Report Viewer, double-click **`View Programming File Generation Report`** in the Process window.

   The Programming File Report contains information about the BitGen run.

For a complete description of BitGen, see the "BitGen" chapter in the *Development System Reference Guide*.

Launching Programming Tools

When you are ready to configure the target device, you need to select a programming tool to use to configure the targeted device. The "Programming Tools" section contains a short overview of each tool.

To launch a programming tool, select the top-level source file in the Source window and then double-click on **`PROM File Formatter`**, **`Hardware Debugger`**, or **`JTAG Programmer`** in the Process window. The selected programming tool opens in its own window with the bitstream file loaded.

Setting Programming File Creation Options

The following sections describe the configuration options you can set prior to creating the programming file. Use the following procedure to access the Process Properties dialog box containing these options.

1. Click on a the top-level design source file in the Source window for a project that targets an FPGA device.
2. Right click on **`Create Programming File`** in the Process window.
3. Select **`Properties`** from the pull-down menu that appears.
4. The Process Properties dialog box for the Create Programming File process appears. An example is shown in the following figure.
5. Click on the tab corresponding to the type of options you want to set to display the available properties. You can set properties for the following program creation option groups: General Options, Configuration Options, Startup Options, and Readback Options.

   Note You can customize whether you want to display the Standard or Advanced list of properties in the Process Properties

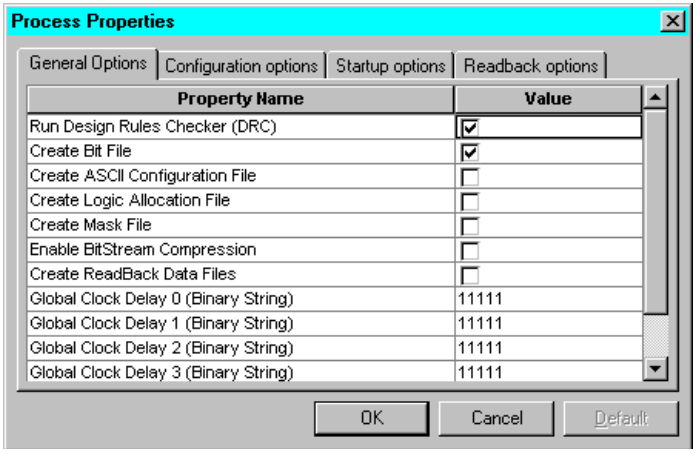
dialog boxes. Use the procedure described in the following section to display the Advanced properties.

Spartan2, Virtex, VirtexE, Virtex2 Options

This section includes descriptions of the available options for the Spartan2, Virtex, VirtexE, and Virtex2 designs. The options listed in each tab vary by architecture. The applicable architectures are identified when an option only applies to certain architectures. In most cases, the Advanced and Standard displays (**`Edit`** → **`Preferences`** → **`Processes`**) are the same. Advanced display options are identified as appropriate.

General Options

The General Option for Program File processing are described in this section. An example of the General Options tab for a Virtex design is shown in the following figure.



Run Design Rules Checker (DRC)

Value: Enabled (default) / Disabled

The Run Design Rules Checker property enables or disables the design rule checker. Before generating the final bitstream, it is important to enable the DRC to evaluate the NCD file for problems that could prevent the design from functioning properly. By default, the Run Design Rules Checker option is enabled (checkbox is checked).

Create Bit File

Value: Enabled (default) / Disabled

The Create Bit File property enables and disables the creation of a design data or bitstream (.bit) file after you have verified the functionality and timing of your placed and routed design.

Create ASCII Configuration File

Value: Disabled (default) / Enabled

The Create ASCII Configuration File property enables and disables the creation of a rawbits text (RBT) file, which is an ASCII representation of your configuration bitstream.

Create Logic Allocation File

Value: Disabled (default) / Enabled

The Create Logic Allocation File property disables and enables the creation of a logic allocation file (*top_source_name*.ll). The Hardware Debugger uses the *top_source_name*.ll file to identify bits in the readback bitstream that represent the values of design I/Os, latches, and flip-flops.

Create Mask File

Value: Disabled (default) / Enabled

The Create Mask File property enables and disables the creation of a mask file (*top_source_name*.msk). The mask file is used to compare relevant bit locations for executing a readback of configuration data contained in an operating FPGA.

Enable BitStream Compression

Value: Disabled (default) / Enabled

The Enable Bitstream Compression property allows you to enable or disable compression of the bitstream file.

Create Readback Data Files

Value: Disabled (default) / Enabled

The Create Readback Data Files property enables and disables the creation of a readback data file.

(Advanced Option) Global Clock Delays (Binary String)

Value: 1111 (default) / Binary String

The **Global Clock Delay 0 (Gclkdel0)**, **Global Clock Delay 1 (Gclkdel1)**, **Global Clock Delay 2 (Gclkdel2)**, and **Global Clock Delay 3 (Gclkdel3)** properties allow you to add delays to the global clocks.

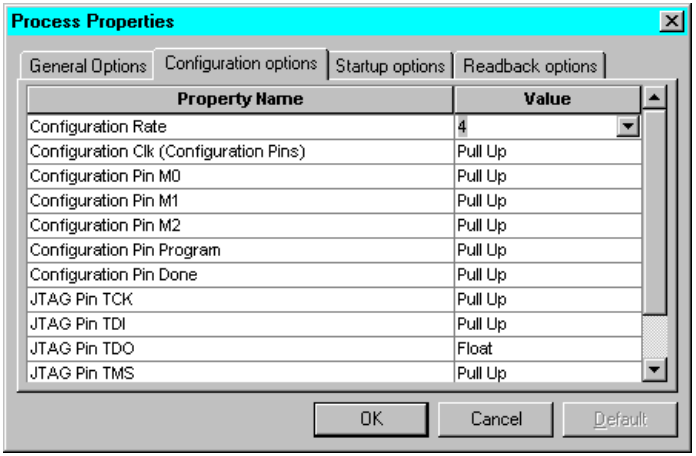
(Advanced Option) Enable Debugging of BitStream

Value: Disabled (default) / Enabled

The Enable Debugging of Bitstream property allows you to enable or disable debugging of the bitstream file.

Configuration Options

The Configuration Options for Program File processing are described in this section. An example of the Configuration Options tab for a Virtex design is shown in the following figure.



Configuration Rate

Value: 4 (default) / 5 / 7 / 8 / 9 / 10 / 13 / 15 / 20 / 26 / 30 / 34 / 41 / 45 / 51 / 55 / 60

Use the configuration rate option to select the rate in megahertz (MHz) for the internal configuration clock, GCLK, when configuring in master mode. The default is 4MHz.

Configuration Clk (Configuration Pins)

Value: Pull Up (default) / Float

The Configuration CLK property allows you to synchronize to an internal clock provided in the FPGA device. Pull Up (the default) enables the pull-up resistor on the Configuration CLK pin. The Float setting disables the pull-up resistor on the Configuration CLK pin.

Configuration Pins

Value: Pull Up (default) / Float / PullDown*

By default, the following configuration pins are set to **`PullUp`** to enable a pull-up on the pin. You can set a configuration pin to **`Float`** to disable both the pull-up and pull-down resistors on the pin or to **`PullDown`**, *except where noted otherwise, to enable a pull-down on the pin.

- Configuration CLK

  This pin can be set to **`PullUp`** or **`Float`** only.

- Configuration Pin M0
- Configuration Pin M1
- Configuration Pin M2
- Configuration Pin Program

  This pin can be set to **`PullUp`** or **`Float`** only.

- Configuration Pin Done

  Select **`Float`** to disable the pull-up resistor on the DONE pin. If you select this option, be sure you have connected an external pull-up resistor on this pin.

  Select **`PullUp`** to enable an internal pull-up resistor on the DONE pin. Select this option only if you do not connect an external pull-up resistor to this pin.

  Select **`Active PullUp`** to drive the DONE pin High with a CMOS driver.

JTAG Pins

Value: Pull Up (default) / Float / PullDown

By default, the following JTAG pins are set to **`PullUp`** to enable a pull-up on the pin. You can set a JTAG pin to **`Float`** to disable both the pull-up and pull-down resistors on the pin or to **`PullDown`** to enable a pull-down on the pin.

JTAG Pin TCK

JTAG Pin TDI

JTAG Pin TDO

JTAG Pin TMS

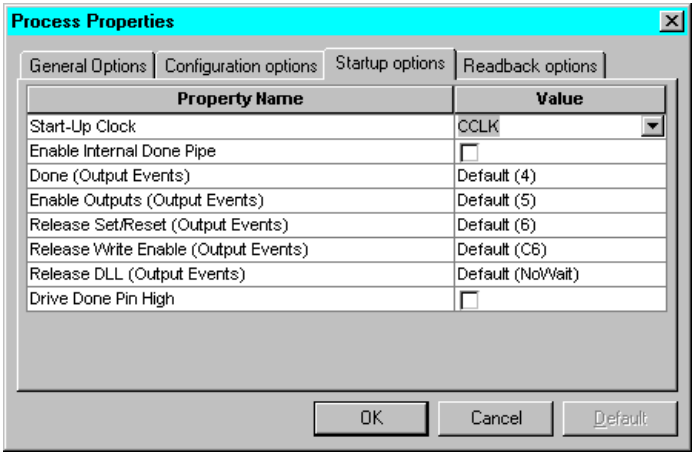
Code (8 Digit hexadecimal)

Value: Blank (default) / 8-digit hexadecimal number

The Code (8 Digit Hexadecimal) property allows you to assign a code in the User Identification Register. Enter an eight-digit hexadecimal ID code in the field. The hexadecimal digits are placed in the User ID Register.

Startup Options

The Startup Option for Program File processing are described in this section. An example of the Startup Options tab for a Virtex design is shown in the following figure.



Start-up Clock

Value: CCLK (default) / User Clock / JTAG Clock

The startup sequence following the configuration of a device can be synchronized to either CCLK, a User Clock, or the JTAG Clock as described below.

- CCLK

  Select CCLK to synchronize to an internal clock provided in the FPGA device.

- User Clock

  Select User Clock to synchronize to a user-defined signal connected to the CLK pin of the startup symbol. You must select this option if your design contains a user clock net that drives the CLK pin on startup.

- JTAG Clock

  Select JTAG Clock to synchronize to the clock provided by JTAG. This clock sequences the TAP controller which provides the control logic for JTAG.

Enable Internal Done Pipe

Value: Disable (default) / Enable

You should enable this option when the startup clock is running at high speeds. If you enable the option, the FPGA waits for the CFG_DONE signal that is delayed by one clock cycle instead of waiting for the pin itself.

Output Events

Value: See Table 16-1

There are five major output events which occur during a device startup.

- Done (CFG_DONE pin going High)
- Enable Outputs (device outputs no longer tri-stated)
- Release Set/Reset (Global Set/Reset signal de-asserted)

- Release Write Enable (Global Write Enable signal de-asserted)
- Release DLL (DLL allowed to synchronize)

Depending on the settings for Startup Clock, the output events can be set to occur as shown in the following table. For more information, see The Programmable Logic Data Book.

Table 16-1 Virtex, VirtexE, Virtex2, Spartan2 Output Events Options Matrix

	CCLK	User Clock	JTAG Clock
DONE	C1-C6	C1, U2-U6	C1, J2-J6
Enable Outputs	C1-C6, Done, Keep	C1, U2-U6, Done, Keep	C1, J2-J6, Done, Keep
Release Set / Reset	C1-C6, Done, Keep	C1, U2-U6, Done, Keep	C1, J2-J6, Done, Keep
Release Write Enable	C1-C6, Done, Keep	C1, U2-U6, Done, Keep	C1, J2-J6, Done, Keep
Release DLL	C0-C6, No Wait	C0-C1, U2-U6, No Wait	C0-C1, J2-J6, No Wait

The definitions of the possible output events settings are as follows.

- C0 — before the Cclk rising edge after the length count is met
- C1 — first-Cclk rising edge after the length count is met
- C2 — second-Cclk rising edge after the length count is met
- C3 — third-Cclk rising edge after the length count is met
- C4 — fourth-Cclk rising edge after the length count is met
- C5 — fifth-Cclk rising edge after the length count is met
- C6 — sixth-Cclk rising edge after the length count is met
- U2 — second-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met)
- U3 — third-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met)
- U4 — fourth-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met)

- U5 — fifth-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met)
- U6 — sixth-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met)
- J2 — second-valid-JTAG-clock rising edge after C1 (first-Cclk rising edge after length count is met)
- J3 — third-valid-JTAG-clock rising edge after C1 (first-Cclk rising edge after length count is met)
- J4 — fourth-valid-JTAG-clock rising edge after C1 (first-Cclk rising edge after length count is met)
- J5 — fifth-valid-JTAG-clock rising edge after C1 (first-Cclk rising edge after length count is met)
- J6 — sixth-valid-JTAG-clock rising edge after C1 (first-Cclk rising edge after length count is met)
- Done — when the CFG_DONE signal goes High
- Keep — holds the pin at whatever level (High or Low) the pin is when the CFG_DONE signal goes High
- No Wait — not synchronized to the startup clock; DLL synchronizes as soon as possible

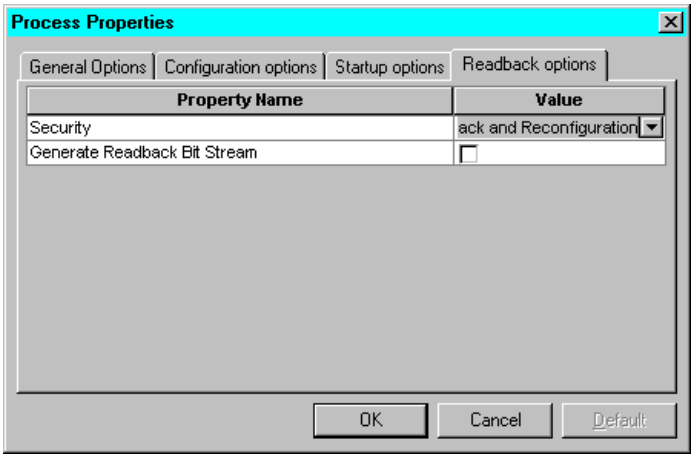
Drive Done Pin High

Value: Disabled (default) / Enabled

The Drive Done Pin High property allows to control when the Done Pin goes High.

Readback Options

The Readback Options for Program File processing are described in this section. An example of the Readback Options tab for a Virtex design is shown in the following figure.



Use this tab to set the following Readback options:

Security

Value: Enable Readback and Reconfiguration (default) / Disable Readback / Disable Readback and Reconfiguration

The Security property allows you to select the readback options. You can set following Readback options from the Security drop-down list box.

- Enable Readback and Reconfiguration

  This option specifies readback options. After the FPGA design has been configured, the FPGA configuration data can be read back and compared with the original configuration data. Readback is initiated by a Low-to-High transition on the M0/RTRIG pin. After this option is run, external logic must drive the Cclk input to read back each data bit. The readback data appears on the $\overline{\text{RDATA}}$ pin.

- Disable Readback

  This option disables readback. Use this option for design security. By disabling readback, configuration data is secure from being read from the FPGA. By default, this option is off.

- Disable Readback and Reconfiguration

  This option disables both readback and reconfiguration. Use this option for design security. By disabling readback and reconfiguration, configuration and reconfiguration data is secure from being read from the FPGA. By default, this option is off.

Generate Readback Bitstreem

Value: Disabled (default) / Enabled

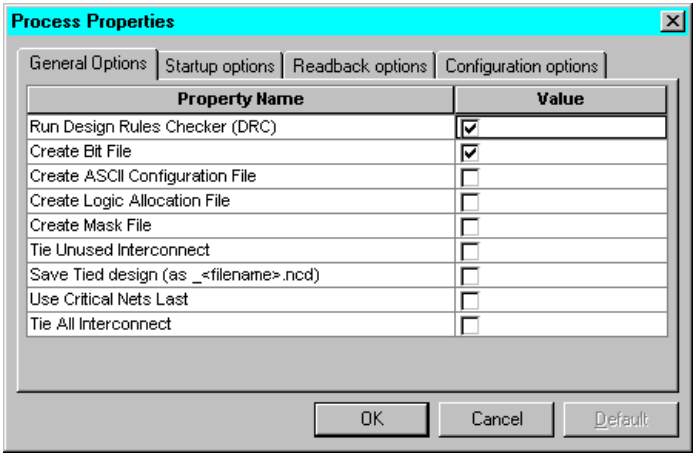
The Generate Readback Bit Stream property allows you to enable the generation of a bitstream. A bitstream file is a stream of data that contains location information for logic on a device, that is, the placement of Configurable Logic Blocks (CLBs), Input/Output Blocks (IOBs), TBUFs, pins, and routing elements. The bitstream also includes empty placeholders that are filled with the logical states sent by the device during a readback. Only the memory elements, such as flip-flops, RAMs, and CLB outputs, are mapped to these placeholders, because their contents are likely to change from one state to another. When downloaded to a device, a bitstream configures the logic of a device and programs the device so that the states of that device can be read back.

Spartan, SpartanXL, XC4000 Options

This section includes descriptions of the available options for the Spartan, SpartanXL, and XC4000 designs. The options listed in each tab vary by architecture. The applicable architectures are identified when an option only applies to certain architectures. In most cases, the Advanced and Standard displays (**`Edit`** → **`Preferences`** → **`Processes`**) are the same. Advanced display options are identified as appropriate.

General Options

The General Options for Program File processing are described in this section. An example of the General Options tab for a Spartan design is shown in the following figure.



Run Design Rules Checker (DRC)

Value: Enabled (default) / Disabled

The Run Design Rules Checker property enables or disables the design rule checker. Before generating the final bitstream, it is important to enable the DRC to evaluate the NCD file for problems that could prevent the design from functioning properly. By default, the Run Design Rules Checker option is enabled (checkbox is checked).

Create Bit File

Value: Enabled (default) / Disabled

The Create Bit File property enables and disables the creation of a design data or bitstream (.bit) file after you have verified the functionality and timing of your placed and routed design.

Create ASCII Configuration File

Value: Disabled (default) / Enabled

The Create ASCII Configuration File property enables and disables the creation of a rawbits text (RBT) file, which is an ASCII representation of your configuration bitstream.

Create Logic Allocation File

Value: Disabled (default) / Enabled

The Create Logic Allocation File property disables and enables the creation of a logic allocation file (*top_source_name*.ll). The Hardware Debugger uses the *top_source_name*.ll file to identify bits in the readback bitstream that represent the values of design I/Os, latches, and flip-flops.

Create Mask File

Value: Disabled (default) / Enabled

The Create Mask File property enables and disables the creation of a mask file (*top_source_name*.msk). The mask file is used to compare relevant bit locations for executing a readback of configuration data contained in an operating FPGA.

Tie Unused Interconnect

Value: Disabled (default) / Enabled

The Tie Unused Interconnect property allows you enable or disable whether all unused interconnects are tied to a logic Low or to a known level, keeping internal noise and power consumption to a minimum. When you enable this option, Design Rule Check (DRC)

runs first. Then, the enabled Tie Unused Interconnect option does the following.

- Ties all possible unused interconnect to unused CLB outputs and configures those outputs with a logic Low (F=0 or G=0)
- Attempts to tie any remaining interconnect to CLB outputs which have not been designated as critical
- Attempts to tie remaining interconnect to the global primary or secondary clock buffer outputs

Save Tied design (as _<filename>.ncd)

Value: Disabled (default) / Enabled

Use the Save Tied design property if you want the tied design saved as _<filename>.ncd.

Use Critical Nets Last

Value: Disabled (default) / Enabled

Enable the Use Critical Nets Last option to use the nets marked as critical to complete the tiedown process if necessary. You should only enable this option as a last resort after an attempt is made to use nets not marked critical.

You can only enable this option if you enabled the Tie all Interconnect option.

Tie All Interconnect

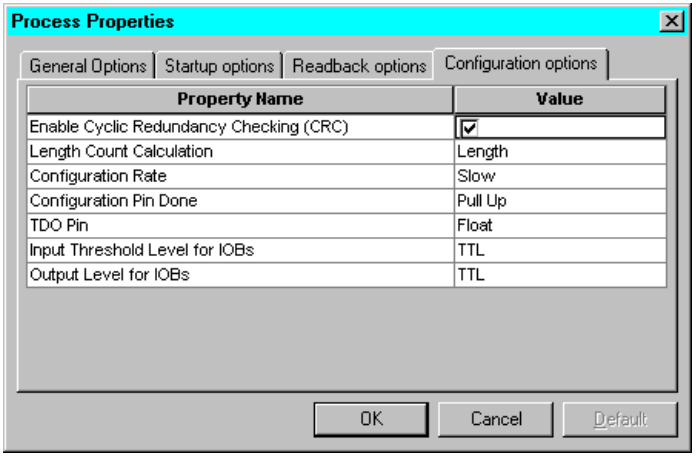
Value: Disabled (default) / Enabled

Enable the Tie All Interconnect option if you want to allow tie down to implement user signals. Enabling this option also forces tie down to fail if all nodes are not tied.

You can only enable this option if you also enable the Tie Unused Interconnect option.

Configuration Options

The Configuration Options for Program File processing are described in this section. An example of the Configuration Options tab for a Spartan design is shown in the following figure.



Enable Cyclic Redundancy Checking (CRC)

Value: Enabled (default) / Disabled

This option enables Cyclic Redundancy Checking (CRC) error checking during configuration. If enabled, the software calculates a running CRC and inserts a unique four-bit partial check at the end of each data frame in the configuration bitstream. This option allows the device to perform a CRC check on the bitstream during the configuration process. If disabled, the device performs a simple check for the 0110 pattern at the end of each frame in the configuration data. By default, this option is on.

Length Count Calculation

Value: Length (default) / DONE

The Length Count Calculation property controls when the device changes from configuration to user operation. The Length Count Alignment and DONE Alignment properties are discussed in *The Programmable Logic Data Book.*

Configuration Rate

Value: Slow (default) / Fast

Use the configuration rate option to select the rate for the internal configuration clock, GCLK, when configuring in master mode. You can set the rate to Slow (1MHz) or Fast (8MHz).

Configuration Pin Done

Value: Pull Up (default) / Float

Use the Configuration Pin Done option as follows:

- Select **`Float`** to disable the pull-up resistor on the DONE pin. If you select this option, be sure you have connected an external pull-up resistor on this pin.
- Select **`PullUp`** to enable an internal pull-up resistor on the DONE pin. Select this option only if you do not connect an external pull-up resistor to this pin.

TDO Pin

Value: Float (default) / Pull Up / Pull Down

The TDO Pin option allows you to enable/disable the pull-up and / or pull down resistor on the TDO pin.The value of the pull-up and pull-down resistors is 50 to 100 kilohms. The following options are available.

- Select Float to disable both the pull-up resistor and pull-down resistor on the TDO pin.
- Select PullUp to enable a pull-up on the TDO pin.
- Select PullDown to enable a pull-down on the TDO pin.

(XC4000 Only) Configuration Pin M0

Value: Float (default) / Pull Up / Pull Down

The M0 pin is used to determine the configuration mode. The value of the pull-up and pull-down resistors is 50 to 100 kilohms. The following options are available.

- Select Float to disable both the pull-up resistor and pull-down resistor on the M0 pin.

- Select PullUp to enable a pull-up on the M0 pin.
- Select PullDown to enable a pull-down on the M0 pin.

(XC4000 Only) Configuration Pin M1

Value: Float (default) / Pull Up / Pull Down

The M1 pin can be used as tri-statable output pin. The value of the pull-up and pull-down resistors is 50 to 100 kilohms. The following options are available.

- Select Float to disable both the pull-up resistor and pull-down resistor on the M1 pin.
- Select PullUp to enable a pull-up on the M1 pin.
- Select PullDown to enable a pull-down on the M1 pin.

(XC4000 Only) Configuration Pin M2

Value: Float (default) / Pull Up / Pull Down

The M2 pin is used to determine the configuration mode. The value of the pull-up and pull-down resistors is 50 to 100 kilohms. The following options are available.

- Select Float to disable both the pull-up resistor and pull-down resistor on the M2 pin.
- Select PullUp to enable a pull-up on the M2 pin.
- Select PullDown to enable a pull-down on the M2 pin.

(XC4000XLA only) Enable Express Mode Bitstream

Value: Disabled (default) / Enabled

This option enables express mode configuration. In this mode, configuration data is presented to the device in parallel format, and each new byte is clocked into the target device with every rising edge of the CCLK. This mode is eight times as fast as other configuration modes because data is processed at the rate of one byte per CCLK rather than one bit per CCLK.

(XC4000XLA only) Tolerate 5V I/O in 3.3V Circuitry

Value: On (default) / Off

This option allows a 3.3V device circuitry to tolerate 5V operation. For any device that operates on a mixed circuit environment with 3.3V and 5V, use this option. For any circuitry that operates exclusively on 3.3V, such as in a laptop computer, turn this option off. Turning off this option reduces power consumption.

Note Disabling this option allows the device's clamping diodes to clamp ringing transients back to the 3.3V supply rail. A clamping diode is connected from each output to VCC. This option affects all I/O pins.

(XC4000XLA only) Enable BSCAN-Based Configuration

Value: Enabled (default) / Disabled

This option allows BSCAN-based configuration after the device is successfully configured. This feature allows board testing without the risk of reconfiguring XLA devices by toggling the TCK/TMS/TDI/TDO lines.

(XC4000XLA only) Allow Direct Sensing of DONE Configuration State (after BSCAN)

Value: Disabled (default) / Enabled

This option allows direct sensing of the DONE configuration state after performing a BSCAN-based configuration. This allows you to determine if a BSCAN-based configuration was successful.

Input Threshold Levels for IOBs

Value: TTL (default) / CMOS / Read from Design

Use the Input Threshold Levels for IOBs option to set one of the following input options:

- Select TTL to specify TTL-compatible inputs.
- Select CMOS to specify CMOS-compatible inputs.
- Select Read from Design to specify the TTL/CMOS input level included in the physical constraints (PCF) file.

Output Level for IOBs

Value: TTL (default) / CMOS / Read from Design

Use the Output Threshold Levels for IOBs option to set one of the following output options:

- Select TTL to specify TTL-compatible inputs.
- Select CMOS to specify CMOS-compatible inputs.
- Select Read from Design to specify the TTL / CMOS input level included in the physical constraints (PCF) file.

(XC4000 Only) Address Lines

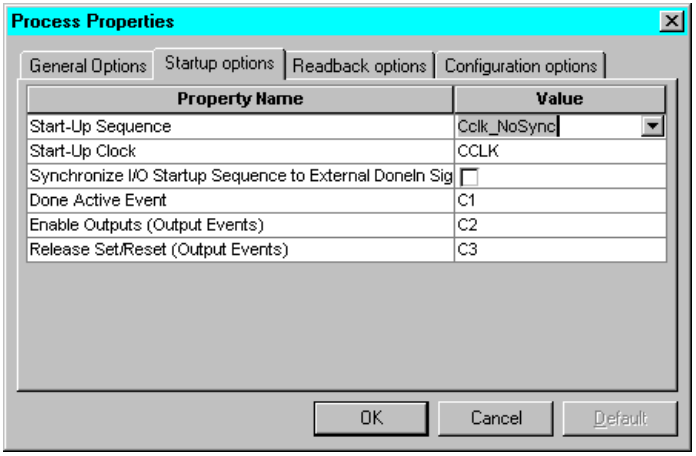
Value: 18 (default) / 22

Use this option to set the number of address lines that will be used by the FPGA during device configuration. Address lines are used to address data from a parallel PROM or flash memory device. Select either 18 or 22. If you choose 22, four extra device pins are activated as configuration address lines.

This option only applies to master parallel mode configuration. You must set this option in addition to setting the mode pins. Refer to *The Programmable Logic Data Book* for more information on address lines and master parallel mode configuration.

Startup Options

The Startup Options for Program File processing are described in this section. An example of the Startup Options tab for a Spartan design is shown in the following figure.



Start-Up Sequence

Value: Cclk_NoSync (default) / Cclk_Sync / Uclk_NoSync / Uclk_Sync

The Start-Up Sequence property allows you to specify the output events to occur according to the setting. By default, this field is set to Cclk_NoSynch. For more information regarding these settings, see *The Programmable Logic Data Book*.

Start-up Clock

Value: CCLK (default) / User Clock / JTAG Clock

The startup sequence following the configuration of a device can be synchronized to either CCLK, a User Clock, or the JTAG Clock. The default is CCLK.

- CCLK

  Select CCLK to synchronize to an internal clock provided in the FPGA device.

- User Clock

  Select User Clock to synchronize to a user-defined signal connected to the CLK pin of the startup symbol. You must select this option if your design contains a user clock net that drives the CLK pin on startup.

- JTAG Clock

  Select JTAG Clock to synchronize to the clock provided by JTAG. This clock sequences the TAP controller which provides the control logic for JTAG.

Synchronize I/O Startup Sequence to External DoneIn Signal

Value: Disabled (default) / Enabled

The startup sequence can be synchronized with the signal on the DONE pin. Enable this option to begin the startup sequence when the signal on the DONE pin goes High. Typically, this option is enabled if the design configures a device connected in a daisy chain. Disable this option to begin the sequence when the configuration memory is full.

Output Events

Value: See Table 16-2

There are three major output events which occur during a device startup.

- **Done Active Event** (DONE pin going High)
- **Enable Outputs** (device outputs no longer tristated)
- **Release Set/Reset** (Global Set/Reset signal deasserted)

Depending on the settings for Start-Up Clock and Synchronize I/O Startup Sequence to External DONE Input Pin, the output events can be set to occur as shown in the following table. For more information, see *The Programmable Logic Data Book.*

Table 16-2 Spartan, SpartanXL, XC4000 Output Events Options Matrix

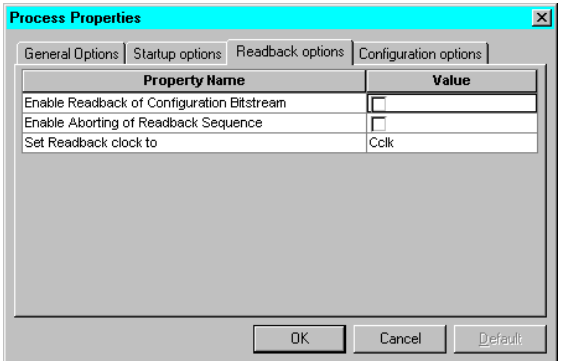
	CCLK Sync	CCLK No Sync	User Clock Sync	User Clock No Sync
DONE	C1-C3	C1-C4	C1, U2	C1, U2-U4
Enable Outputs	C2, C3, DI, DI+1	C2-C4	U2, DI, DI+1, DI+2	U2-U4
Release Set/Reset	C2, C3, DI, DI+1	C2-C4	U2, DI, DI+1, DI+2	U2-U4

The definitions of the possible output events settings are as follows.

- C1 — First-Cclk rising edge after the length count is met
- C2 — Second-Cclk rising edge after the length count is met
- C3 — Third-Cclk rising edge after the length count is met
- C4 — Fourth-Cclk rising edge after the length count is met
- U2 — Second-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met)
- U3 — Third-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met)
- U4 — Fourth-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met)
- DI — When the DoneIn signal goes High
- DI+1 — First-Cclk or valid-user-clock rising edge, depending on the selection of start-upClk, after DoneIn goes High
- DI+2 — Second-Cclk or valid-user-clock rising edge, depending on the selection of start-upClk, after DoneIn goes High

Readback Options

The Readback Options for Program File processing are described in this section. An example of the Readback Options tab for a Spartan design is shown in the following figure.



Enable Readback of Configuration Bitstream

Value: Disabled (default) / Enabled

Use this option to enable or disable the readback capability of the configuration bitstream. To enable the readback capability, you enable this option and include the READBACK symbol in your design. Enabling this option generates a .ll file.

Enable Aborting of Readback Sequence

Value: Disabled (default) / Enabled

Enable this option to abort the readback sequence. You can use this option to terminate the readback sequence when the device detects a High-to-Low transition on the TRIG pin of the READBACK symbol.

Set Readback clock to

Value: Cclk (default) / Rdbk

During the readback process, the data is clocked out synchronously. The source of the readback clock can be either the CCLK or a user clock. Select the Cclk value to clock out the readback data through an internal clock provided in the FPGA device. Select the Rdbk value to out the readback data through a user-defined signal connected to the CLK pin of the READBACK symbol.

Creating 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 JTAG Programmer uses this JED file to configure XC9500 / XL / XV CPLD devices.

To create a JED programming file for your design, use the following procedure.

1. Select the top-level source for the project in the Source window.
2. Click **`Create Programming File`** in the Process window.
3. Click **`Process`** → **`Run`** in the Project Navigator menu. (An alternative method is to double-click on **`Creating Programming File`** in the Process window.)
4. The programming file creation process runs. If there are no errors, the *top_source_name*.jed file is created.

Launching the JTAG Programmer

When you are ready to configure the target device, you need to use the JTAG Programmer to configure the targeted device. The "Programming Tools" section contains a short overview of the JTAG Programmer.

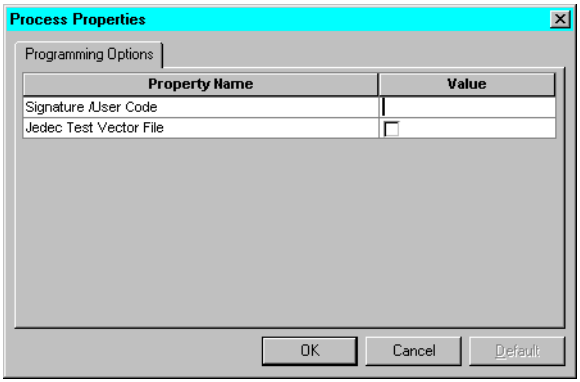
To launch the JTAG Programmer, select the top-level source file in the Source window and then double-click on **`JTAG Programmer`** in the Process window. The TAG Programmer opens in its own window with the JED file loaded.

Setting Programming File Creation Options

This section describes the programming options you can set prior to creating the programming file. Use the following procedure to access the Process Properties dialog box containing these options.

1. Click on a the top-level design source file in the Source window for a project that targets an FPGA device.
2. Right click on **`Create Programming File`** in the Process window.
3. Select **`Properties`** from the pulldown menu that appears.

4. The Process Properties dialog box (shown in the following figure) for the Create Programming File process appears.



You can set the following options for CPLD programming file creation:

- Signature / User Code

  Value: Blank (default) / Four-character text string

  Enter 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 JTAG Programmer to identify the configuration data signature (user-code) of a programmed XC9500 or XC9500XL device. The default is to use the *top_source_name*.

- Jedec Test Vector File

  Value: Disabled (default) / Enabled

  Enable this option to include 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.

Programming Tools

After the programming file has been successfully created, you can use one of the programming tools described in this section to configure your device.

JTAG Programmer

The JTAG Programmer downloads, reads back, and verifies FPGA and CPLD design configuration data. It can also perform functional tests on any device and probe the internal logic states of your design.

The JTAG Programmer software can be used to configure both FPGAs and CPLDs and supports both the XChecker and the Parallel Cable III. This is a GUI based program. See the *JTAG Programmer Guide* for details. Also, see the *Hardware User Guide* for information about cable compatibility.

PROM File Formatter

A 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 chai of FPGAs through a microprocessor. The file is sotred 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 do the following.

- 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

Hardware Debugger

The Hardware Debugger is a graphical interface that allows you to download an FPGA design to a device, verify the downloaded configuration, and display the internal states of the programmed device.

The Hardware debugger can download a BIT file or a PROM file: MCS, EXO, or TEK file formats. A BIT file contains configuration information for an FPGA device. Form more information on using the Hardware Debugger, see the *Hardware Debugger Guide*.