# Contents

What’s New in This Release ........................................... xiv
About This Guide......................................................... xvii
Customer Support......................................................... xxi

1. Introduction

   About the FPGA Tool ............................................. 1-2
   Features and Benefits ............................................ 1-2
   Methodology ....................................................... 1-3

2. Choosing a Design Flow

   DesignWizard Flow .............................................. 2-2
   Push-Button Flow ................................................ 2-3
   Performance (Constraint-Driven) Flow ......................... 2-4
   Script-Based Flow ............................................... 2-4
   Hierarchical (Multiple-Device) Flow ......................... 2-5
3. Running the FPGA Tool From the GUI or the Shell
   Using the Graphical User Interface ........................................... 3-2
   About the Tip Bar ................................................................. 3-2
   About the Output Window ....................................................... 3-2
   Using the Mouse Buttons and Context Menus. ......................... 3-2
   Getting Help from the GUI ...................................................... 3-3
   Using the Shell ........................................................................... 3-4
   FPGA Scripting Tool (FST) Command Overview ...................... 3-5
       Command Groups ................................................................... 3-6
   Using the Man Pages ................................................................. 3-7

4. Creating Design Source Files
   Using Memory Elements With Lucent Devices ......................... 4-3
   Mixing HDL and Netlist Input ..................................................... 4-7
   Creating a Finite State Machine (FSM) ................................. 4-9
       Creating an FSM Using a VHDL Template .......................... 4-9
       Creating an FSM Using a Verilog Template ..................... 4-12
   Using Encrypted Intellectual Property .................................. 4-14

5. Setting Up Projects
   Creating a Project ................................................................. 5-2
   Specifying Design Source Files .............................................. 5-3
   Debugging Design Source Files .............................................. 5-6
6. Elaborating Designs
   Elaborating a Design .................................................. 6-2
   Identifying the Top-Level Design ................................. 6-2
   Creating the Implementation ....................................... 6-3

7. Setting Constraints, Attributes, and Controls
   Importing Constraints ................................................ 7-3
   Entering Constraints, Attributes, and Controls ............... 7-3
     Editing Table Cells .................................................. 7-4
   Setting Multicycle Timing Constraints ......................... 7-5

8. Optimizing Design Implementations
   Setting General Synthesis Options ............................... 8-3
   Setting Project Synthesis Options ............................... 8-5
   Updating and Force-Updating a Project After Incremental Changes 8-9

9. Viewing Schematics
   Viewing a Schematic of an RTL (Generic) Design ............ 9-4
   Viewing a Schematic of an Optimized (Mapped) Design ...... 9-5

10. Viewing and Debugging Timing Results
    Analyzing Timing ...................................................... 10-2
    Viewing Timing Results Graphically Using TimeTracker .... 10-5
11. Generating Netlists, Reports, Project Databases, and Using Quartus II

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generating a Netlist</td>
<td>11-2</td>
</tr>
<tr>
<td>Generating a Report</td>
<td>11-3</td>
</tr>
<tr>
<td>Generating a Project Database</td>
<td>11-5</td>
</tr>
<tr>
<td>Using Quartus II</td>
<td>11-6</td>
</tr>
</tbody>
</table>

12. Using Constraint Files

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>About the ASCII Constraint File Format</td>
<td>12-3</td>
</tr>
<tr>
<td>Header Information</td>
<td>12-4</td>
</tr>
<tr>
<td>Clock Constraints</td>
<td>12-4</td>
</tr>
<tr>
<td>Object Groups</td>
<td>12-5</td>
</tr>
<tr>
<td>Path Constraints</td>
<td>12-5</td>
</tr>
<tr>
<td>Port Constraints</td>
<td>12-7</td>
</tr>
<tr>
<td>Module Constraints</td>
<td>12-8</td>
</tr>
<tr>
<td>Register Constraints</td>
<td>12-9</td>
</tr>
<tr>
<td>Vendor Options</td>
<td>12-9</td>
</tr>
<tr>
<td>Exporting an ASCII Constraint File</td>
<td>12-10</td>
</tr>
<tr>
<td>Importing an ASCII Constraint File</td>
<td>12-11</td>
</tr>
</tbody>
</table>

Appendix A. FPGA Scripting Tool (FST)

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>A-2</td>
</tr>
<tr>
<td>Project Variables</td>
<td>A-3</td>
</tr>
<tr>
<td>FST Commands</td>
<td>A-3</td>
</tr>
<tr>
<td>Constraints Commands</td>
<td>A-3</td>
</tr>
<tr>
<td>Command Type</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Reports Commands</td>
<td>A-4</td>
</tr>
<tr>
<td>Browsing Objects Commands</td>
<td>A-5</td>
</tr>
<tr>
<td>Timing Commands</td>
<td>A-5</td>
</tr>
<tr>
<td>Source Design Management Commands</td>
<td>A-5</td>
</tr>
<tr>
<td>Target Management Commands</td>
<td>A-6</td>
</tr>
<tr>
<td>Logical Library Management Commands</td>
<td>A-6</td>
</tr>
<tr>
<td>Chip Management Commands</td>
<td>A-6</td>
</tr>
<tr>
<td>Source File Management Commands</td>
<td>A-7</td>
</tr>
<tr>
<td>Project Management Commands</td>
<td>A-7</td>
</tr>
<tr>
<td>Built-In Tcl Commands</td>
<td>A-7</td>
</tr>
<tr>
<td>Other Commands</td>
<td>A-9</td>
</tr>
</tbody>
</table>

**Glossary**

**Index**
## Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1-1</td>
<td>FPGA Express in Your Design Environment.</td>
<td>1-5</td>
</tr>
<tr>
<td>Figure 5-1</td>
<td>The Project Window After You Add Design Source Files</td>
<td>5-5</td>
</tr>
<tr>
<td>Figure 5-2</td>
<td>HDL Editor Window</td>
<td>5-7</td>
</tr>
<tr>
<td>Figure 6-1</td>
<td>Drop-Down List of Top-Level Designs in the Toolbar</td>
<td>6-2</td>
</tr>
<tr>
<td>Figure 6-2</td>
<td>Create Implementation Dialog Box</td>
<td>6-3</td>
</tr>
<tr>
<td>Figure 6-3</td>
<td>Creating the Design Implementation</td>
<td>6-5</td>
</tr>
<tr>
<td>Figure 7-1</td>
<td>Constraint Tables</td>
<td>7-3</td>
</tr>
<tr>
<td>Figure 7-2</td>
<td>Shift Register With Multicycle Timing Constraint Assignment</td>
<td>7-5</td>
</tr>
<tr>
<td>Figure 7-3</td>
<td>Create/Edit Timing Sub Path Window</td>
<td>7-7</td>
</tr>
<tr>
<td>Figure 7-4</td>
<td>Startpoints and Endpoints for New Subpath</td>
<td>7-8</td>
</tr>
<tr>
<td>Figure 7-5</td>
<td>New Subpath (With Multicycle Timing Constraint) in Paths Constraint Table</td>
<td>7-9</td>
</tr>
<tr>
<td>Figure 8-1</td>
<td>General Options</td>
<td>8-3</td>
</tr>
<tr>
<td>Figure 8-2</td>
<td>Project Options</td>
<td>8-6</td>
</tr>
<tr>
<td>Figure 9-1</td>
<td>RTL Version of a Design in the Schematic Viewer</td>
<td>9-2</td>
</tr>
</tbody>
</table>
Figure 9-2 Optimized (Mapped) Version of a Design in the Schematic Viewer ........................................... 9-3
Figure 10-1 Optimization Results in the Clocks Constraint Table . . . 10-2
Figure 10-2 Optimization Results in the Paths Constraint Table ....... 10-3
Figure 10-3 Optimization Results in the Ports Constraint Table...... 10-4
Figure 10-4 Optimization Results in the Modules Constraint Table . . 10-5
Figure 11-1 Export Netlist Dialog Box ................................................. 11-2
Figure 11-2 Generate Project Report Dialog Box ......................... 11-5
Figure 11-3 Export Netlist Dialog Box ................................................. 11-6
Figure 11-4 Place and Route Dialog Box ............................ 11-7
Tables

Table 3-1  Mouse Behavior in FPGA Express .......................... 3-2
Preface

This preface includes the following sections:

• What's New in This Release
• About This Guide
• Customer Support
What’s New in This Release

This section describes the new features and enhancements in FPGA Express version 3.6 and 3.6.1. Unless otherwise noted, you can find more information about these changes later in this book.

New Features

FPGA Express version 3.6 and 3.6.1 includes the following new features.

- For Xilinx Virtex-II input, output, triout and inout pad cells, you can use the Edit Constraints > Ports tab to attribute the ports on the design with a specific combination of the IO Standard, Slew Rate and Drive Strength attributes. Depending on these attributes, FPGA Express infers the appropriate Virtex-II primitives, and the pad mapper maps to a pad cell with the attributes that you specify.

  For more information about setting these attributes, see the online Help.

  In addition to the GUI, you can use shell commands for this feature. For more information, see the man pages.

- With version 3.6 or later, you can install multiple versions of FPGA Express or any combination of the Synopsys FPGA products on a single machine.

  For details about installing FPGA Express, see the FPGA Express Installation Guide.
• You can change the order in which the FPGA tool analyzes source files after you add them to a design. Also, you can use the Auto-Analyze Added Files check box (which is in the General tab of the synthesis Options dialog box) to disable automatic analysis of source files as you add them to a design.

For information about using these features, see the online Help.

• The schematic timing path browser displays critical timing paths after you analyze synthesized circuits. It includes a third schematic window specifically for viewing a selected timing path. The browser generates a flattened representation of elements on the critical path and any intermediate hierarchical boundaries.

For information about using this feature, see the online Help.

---

**Enhancements**

FPGA *Express* version 3.6 and 3.6.1 includes the following enhancements.

• The HDL Editor contains several enhancements:
  - Color codes regular text, selected (user-highlighted) text, numbers, operators, comments, keywords, strings, and the HDL Editor background; you can change these settings
  - Supports multiple undo and redo
  - Automatically indents lines as you create them (and lets you disable automatic line indentation)
  - Indicates when you have changed the current file since you saved it
  - Displays the line number and column number on the status bar
Also, the HDL Editor lets you

- Jump to a specific line number
- Suppress the display of line numbers in the design source file
- Display tabs and define tab size
- Search for text in the upward direction (in addition to the downward direction)
- Replace text interactively or globally

For more information about these enhancements, see the online Help.

- You can define attributes in Verilog source files and pass them to the New HDL Compiler (Presto). The New HDL Compiler uses these attributes for internal optimization and also passes them (as property statements in an EDIF netlist) to place and route tools.

The input format and the output format are unchanged from the existing HDL compiler.

For more information about this enhancement, see the online Help.

- This release adds the DIV (divider) arithmetic operator.
Known Limitations and Workarounds

Information about known problems, limitations, and workarounds is in the FPGA Express Release Note, which is available from the Help menu in the GUI.

To view the Release Note,

- Choose Help > Release Note.

The Release Notes are also available in SolvNet at the Synopsys website. For more information, see “Accessing SolvNet” on page xxi.

About This Guide

This guide provides information for users of FPGA Express. The guide explains how to apply the program’s basic features to your chosen design flow and FPGA architecture. For information about features not covered in this guide, see the online Help or man pages for individual commands.

Audience

This guide is for logic designers or engineers who use the FPGA Express synthesis tool to implement FPGA designs.
Related Publications

For additional information about FPGA Express, see

- *FPGA Express Installation Guide* (delivered with the program in portable document format (PDF) files)
- *FPGA Compiler II / Express VHDL Reference Manual* (online) (delivered with the program in portable document format (PDF) files)
- *FPGA Compiler II / Express Verilog HDL Reference Manual* (online) (delivered with the program in portable document format (PDF) files)
- Synopsys Online Documentation (SOLD), which is included with the software
- Documentation on the Web, which is available through SolvNET at http://solvnet.synopsys.com
- The Synopsys MediaDocs Shop, from which you can order printed copies of Synopsys documents, at http://mediadocs.synopsys.com
- The program’s Help menu, which provides access to a selection of FPGA-vendor documents from FPGA vendors.
- Vendor-provided documents: A selection of documents provided by FPGA vendors is available from the Help menu. You might find these documents helpful in answering your vendor-specific questions.
• Compiler reference manuals: Online compiler reference manuals for VHDL and Verilog HDL contain recommended coding styles and their application to FPGA Express synthesis and optimization. You can find all compiler reference manuals on the FPGA Express CD-ROM or by clicking the Help menu.
## Conventions

The following conventions are used in Synopsys documentation.

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Courier</td>
<td>Indicates command syntax.</td>
</tr>
<tr>
<td><strong>Courier italic</strong></td>
<td>Indicates a user-defined value in Synopsys syntax, such as <code>object_name</code>. (A user-defined value that is not Synopsys syntax, such as a user-defined value in a Verilog or VHDL statement, is indicated by regular text font italic.)</td>
</tr>
<tr>
<td><strong>Courier bold</strong></td>
<td>Indicates user input—text you type verbatim—in Synopsys syntax and examples. (User input that is not Synopsys syntax, such as a user name or password you enter in a GUI, is indicated by regular text font bold.)</td>
</tr>
<tr>
<td>[]</td>
<td>Denotes optional parameters, such as <code>pin1 [pin2 ... pinN]</code></td>
</tr>
<tr>
<td></td>
<td>Indicates a choice among alternatives, such as `low</td>
</tr>
<tr>
<td>_</td>
<td>Connects terms that are read as a single term by the system, such as <code>set_annotated_delay</code></td>
</tr>
<tr>
<td>Control-c</td>
<td>Indicates a keyboard combination, such as holding down the Control key and pressing c.</td>
</tr>
<tr>
<td>\</td>
<td>Indicates a continuation of a command line.</td>
</tr>
<tr>
<td>/</td>
<td>Indicates levels of directory structure.</td>
</tr>
<tr>
<td><strong>Edit &gt; Copy</strong></td>
<td>Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.</td>
</tr>
</tbody>
</table>
Customer Support

You can access customer support through SolvNet online customer support or through the Synopsys Technical Support Center.

Accessing SolvNet

SolvNet includes the Solv-It electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. Also, SolvNet gives you access to a wide range of Synopsys online services including software downloads, documentation on the Web, and “Enter a Call With the Support Center.”

To access SolvNet,


2. If prompted, enter your user name and password. (If you do not have a Synopsys user name and password, click New Synopsys User Registration.)

If you need help using SolvNet, click SolvNet Help in the column on the left side of the SolvNet Web page.
Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways.

- Open a call to your local support center from the Web by going to http://solvnet.synopsys.com (you need a Synopsys user name and password). Then click “Enter a Call With the Support Center.”

- Send an e-mail message to support_center@synopsys.com

- Telephone your local support center.
  - From the continental United States, call (800) 245-8005.
  - From Canada, call (650) 584-4200.
  - Find other local support center telephone numbers at http://www.synopsys.com/support/support_ctr.

For more information about FPGA Express, see the FPGA Web page at http://www.synopsys.com/products/fpga.
1

Introduction

FPGA Express is the FPGA logic-synthesis solution from Synopsys, Inc. FPGA Express brings a combination of synthesis and optimization technology, high-level design methodology, and easy-to-use interfaces to the FPGA design desktop.

This chapter contains the following sections:

- About the FPGA Tool
- Features and Benefits
- Methodology
About the FPGA Tool

FPGA Express is a synthesis tool for FPGA architectures. With the tool, you can create optimized netlists from VHDL code, from Verilog HDL code, or from existing unoptimized EDIF or Xilinx Netlist Format (XNF) netlists.

Features and Benefits

The FPGA Express core technology was developed specifically for FPGA architectures. The technology includes the following features:

- Architecture-specific mapping and optimization for leading programmable logic vendors
- Industry-leading quality of results (QOR)
- Block-Level Incremental Synthesis (BLIS) for Xilinx Virtex devices.
- Integration with Quartus
- Support for industry-standard HDL
- Easy-to-use design flows and graphical user interfaces
- Integrated static timing analysis with TimeTracker
- Schematic viewing with tight links to TimeTracker
- Tcl-based language for scripting
If you are migrating from a schematic-based methodology to an HDL-based methodology, FPGA Express adds HDL logic synthesis and optimization to your FPGA design environment. You can define a design completely with HDL source code or use a mixture of schematics and HDL source code to enter a design into FPGA Express.

Using an HDL-based design methodology increases productivity, because HDL source code is vendor-independent, reusable, and you can retarget it toward various technologies. The FPGA Express optimization algorithms are specifically tuned for each FPGA vendor’s device family.

The FPGA Express GUI lets you use standard commands and procedures to input values. It does not require command scripts. Also, it allows multiple synthesis design flows, which lets you use a variety of design styles and performance goals.

## Methodology

Your specific methodology determines how you use FPGA Express in your design flow. However, the following general methodology explains how FPGA Express fits best into your design flow.

You create and validate your design before you use FPGA Express. You can use any of the following design creation methods:

- For an HDL-based design methodology, you write the HDL source code for the design.
- For a schematic-based methodology, you capture the design schematics and export them to an HDL format, netlist format, or a combination of these.
• For a mixed (HDL and schematic) design methodology, you write the HDL source code for the parts of the design you want to describe in HDL. You capture the schematics for the rest of the design.

If you want to verify the design functionality, you use an HDL simulator for HDL code and use a gate-level simulator for schematics.

Figure 1-1 shows the flow of FPGA Express with other tools in these design environments. The steps in this procedure are similar for each environment; only the method of entering the design into FPGA Express differs. Note that simulation is optional.
Figure 1-1 FPGA Express in Your Design Environment

Introduction: Methodology
You use the following general procedure when using FPGA Express. The way you enter the design into FPGA Compiler II differs for each method described previously, but the steps in this procedure are similar. Simulation is optional:

1. Create design source files.

2. Launch FPGA Express.

3. Set up the design project and analyze the design source files.
   
   This step creates an FPGA Express project. The project is a working directory that holds internal and intermediate files. See Chapter 5, “Setting Up Projects.”

4. Elaborate the design.

   In this step, you identify the top-level design, choose the target device, set the clock frequency, enter any constraints, and create the implementation. See Chapter 6, “Elaborating Designs.”

   The tool elaborates each subdesign module as it creates and links the design hierarchy. The resulting implementation is a generic gate-level representation of your design.

   After elaboration, the tool generates a schematic that represents the hierarchy of your design. (If the design contains encrypted IP, some tool features are disabled to protect the integrity of the IP. For more information about disabled features, see “Using Encrypted Intellectual Property” on page 4-14.)

5. Optimize the design implementation.

   During optimization, the tool uses architecture-specific algorithms to analyze the timing of your design, compare timing to your requirements, and make changes to meet your design goals. See Chapter 8, “Optimizing Design Implementations.”
Also, the tool can generate reports and an optimized schematic.

6. **(Optional) Analyze timing information to verify circuit performance.**

   The tool displays timing information in tables alongside your design constraints and highlights timing violations. The tool links timing information directly to the schematic for debugging. See Chapter 9, “Viewing Schematics.”

7. **Export an optimized EDIF or XNF netlist for place and route.**

   The tool can generate an optimized netlist ready for place and route using FPGA vendor tools. See “Generating a Netlist” on page 11-2.

8. **(Optional) Generate VHDL or Verilog netlists for functional simulation.**

   By using a standard HDL simulator, you can verify that the optimized netlist is functionally identical to the RTL input. See “Generating a Netlist” on page 11-2.

To finish the design process, complete the following steps outside of FPGA Express:

1. **Place and route the design by using the FPGA vendor’s development system.**

2. **(Optional) Simulate the design with post-place and route timing delays.**

3. **Program the FPGA device.**
Introduction: Methodology
1-8
Choosing a Design Flow

This chapter introduces you to many of the basics of running FPGA Express. It includes the following sections:

- DesignWizard Flow
- Push-Button Flow
- Performance (Constraint-Driven) Flow
- Script-Based Flow
- Hierarchical (Multiple-Device) Flow
- Actel Device Flow

FPGA Express supports a variety of design flows. You choose the flow that meets your design’s requirements.
DesignWizard Flow

The DesignWizard leads you from creating a project to optimizing your chip in as few as two steps.

There are two ways to access DesignWizard:

- When you first start the tool, the DesignWizard starts automatically.
- You can start the DesignWizard manually by choosing DesignWizard from the File menu.

The first DesignWizard screen (FPGA Project dialog box) lets you

- Create a new project or open an existing one
- Disable the DesignWizard for subsequent starts of the tool

The default directory in which the DesignWizard stores the tool outputs is at the same hierarchical level as the file for your top-level design. The default base name of the default directory is identical to that of the top-level design.

The DesignWizard follows this flow:

1. Add design source files and specify the top-level design.
2. Set synthesis parameters.
   - Select target technology
   - (Optional) Set optimization parameters such as Clock frequency, CPU effort, and whether to preserve or eliminate hierarchy.
- (Optional) Change one or more of the following: project name, location in which the DesignWizard stores tool outputs, and various netlist settings.

3. Click Run (Finish).

The DesignWizard stores the tool outputs in a default directory. The default directory is at the same hierarchical level as the file for your top-level design. The default base name of the default directory is identical to that of the top-level design.

At the end of the DesignWizard flow, all results are reflected in the FPGA Express GUI, and you can work on them as usual there. For a more detailed design flow, including setting optimization constraints, see “Performance (Constraint-Driven) Design Flow” in the online Help.

---

**Push-Button Flow**

Use a push-button flow when quick results are more important than detailed design goals. A push-button flow skips the step of setting or adjusting design constraints.

In only two steps, you can go from an HDL description or schematic netlist to an optimized netlist ready for placing and routing. The steps are simple: First, add your design’s source files to the FPGA Express project. Then select the target device (vendor, family, package, and speed grade).

To use the push-button flow, select the Skip constraint entry option in the Create Implementation dialog box (this is the default setting). FPGA Express elaborates and optimizes your design in one step.
Performance (Constraint-Driven) Flow

Use a constraint-driven flow when QOR is most important or when you want to refine your design goals.

With FPGA Express, you can specify design requirements and constraints to tailor the results to your design goals.

To use a constraint-driven flow, deselect the Skip Constraint Entry option in the Create Implementation dialog box. FPGA Express creates just the elaborated implementation, thus allowing you to specify constraints before you optimize the implementation.

Script-Based Flow

You can access all of the functionality of the FPGA Express via the FPGA scripting tool and the FPGA Express Tcl-based command-line language. You can use the scripting tool to synthesize any designs, either in batch mode or interactively, from the shell command line.

Use a script-based flow if

- You prefer command-line or keyboard-based interfaces to graphical interfaces.
- You understand the capabilities of the tool fully and want to specify operations as quickly as possible.

For example, an action that requires multiple mouse clicks and keyboard entries using the graphical interface could be entered as a single command.
For details on running the scripting tool, see “Using the Shell” on page 3-4

Hierarchical (Multiple-Device) Flow

You can use the multiple-device design flow when you want to compare the performance of a design for multiple-device families and vendors. You can manage and analyze different implementations within the same FPGA Compiler II project. The design source files, constraints, and optimization controls are the same; only the target device is different.

You can perform a multiple-device design flow by using these steps.

1. Set up the project.
2. Select the target architecture and elaborate the design.
3. Enter constraints and controls.
4. Optimize logic, generate reports, and export the design constraint file.
5. Analyze timing information.
6. Create a new implementation.
7. Import the design constraint file.
8. Repeat the optimization process and generate netlists and reports.
Choosing a Design Flow: Hierarchical (Multiple-Device) Flow
2-6
Choosing a Design Flow: Hierarchical (Multiple-Device) Flow

2-8
This chapter describes how to use the graphical user interface and the command-line interface in the FPGA tool.

This chapter includes the following sections:

- **Using the Graphical User Interface**
- **Using the Shell**
Using the Graphical User Interface

To launch FPGA Express, choose Start > Programs > Synopsys > FPGA Express from the Windows Start menu.

About the Tip Bar

The tip bar provides information at each stage of the design flow. It automatically detects the state of the design, identifies the next logical step in the design flow, and provides a brief explanation of the function and purpose of that step.

About the Output Window

The Output window displays errors, warnings, and other messages about your actions as you use FPGA Express.

Using the Mouse Buttons and Context Menus

Table 3-1 provides information to help you use FPGA Express more efficiently.

Table 3-1  Mouse Behavior in FPGA Express

<table>
<thead>
<tr>
<th>Action</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left-click</td>
<td>Selects and deselects.</td>
</tr>
<tr>
<td>Left-double-click</td>
<td>In the Design Sources and Chips windows, expands or contracts the hierarchy of the project, library, or design source file.</td>
</tr>
<tr>
<td></td>
<td>On an error in the Output window, starts the text editor and opens the design source file to the selected error.</td>
</tr>
</tbody>
</table>
Table 3-1  Mouse Behavior in FPGA Express (Continued)

<table>
<thead>
<tr>
<th>Action</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Right-click</td>
<td>Displays a context menu of actions for</td>
</tr>
<tr>
<td></td>
<td>- The design source file in the Design Sources</td>
</tr>
<tr>
<td></td>
<td>window and the editor</td>
</tr>
<tr>
<td></td>
<td>- Unoptimized or optimized implementations in the Chips window</td>
</tr>
<tr>
<td></td>
<td>- The messages in the Output window</td>
</tr>
<tr>
<td></td>
<td>- Subpaths in the Paths constraint table</td>
</tr>
</tbody>
</table>

Getting Help from the GUI

FPGA Express includes these forms of online Help:

- Topic-based
- Context-sensitive

To access the topic-based online Help system, choose Help Topics from the Help menu. You can then access specific Help topics through the table of contents, the alphabetic index, or the text-search feature. The Help system also includes a glossary of FPGA Express terminology.

When working in FPGA Express, you can display a context-sensitive description of a specific feature. Click the question mark button on the product’s toolbar or press Shift-F1 and then click an object on the screen.
Using the Shell

You can run scripts from the command line of the FPGA shell, or execute batch files of script commands. The FPGA scripting tool (FST) is a Tcl-based command-line interface to all of the synthesis and optimization features of FPGA. For more information, see Appendix A, “FPGA Scripting Tool (FST).”

To run the scripting tool,

1. Start the shell either by choosing Start > Programs > Synopsys > FPGA Express Shell in Windows or by typing fe_shell in a UNIX shell or command window.

2. In the fe_shell, execute script commands. All of the commands are documented in the man pages.
   - To obtain a list of all available commands, at the shell prompt enter
     
     help
   
   - To access the man pages, enter
     
     man commandname
   
   - To display Help about the man pages, enter
     
     man help

Note:
You can find sample scripts in the samples/Synopsys subdirectory of the FPGA installation.
FPGA Scripting Tool (FST) Command Overview

FST scripting tool commands are in the form

<action>_<object>_<modifier>

The following objects defined by the data model are implemented in FST:

- project
- library
- file
- chip
- module
- clock
- group
- path group

In addition, the following netlist objects are implemented in FST:

- design
- port
- padcell
- net
- pin
Command Groups

Commands are organized into command groups, either by action or by object. A special built-in command group contains a list of supported Tcl keywords, as well as some generic command-line utilities common to Synopsys command-line interfaces.

Commands and options that are not in the built-in command group can be abbreviated as long as the abbreviation is unique. If the abbreviation is not unique, a list of matching commands is displayed.

FST Command Requirements. Most FST commands require a project to be opened. Once a project is open, it is referred to as the current project.

Commands that operate on chips usually operate on the current chip. The current chip is set by the current_chip command.

Modifications to the current chip are generally not saved until you change the current chip or close the current project. The exceptions to this rule are commands that take a chip name as an argument.

For example, some commands, such as export_constraint or export_netlist, export information from a chip to a file. These commands require a chip as an argument and will read a chip before it writes the necessary information. If the chip to be exported is the current chip, it is automatically saved to disk. This behavior ensures that a file correctly reflects the state of a chip in a project.
Using the Man Pages

The man pages contain the most up-to-date command usage and syntax information. To access the man pages, enter `man`, followed by the command name, in the FPGA Express shell (fe_shell). For example,

```
fe_shell> man script_chip
```
Creating Design Source Files

This chapter describes how to create design source files for use with FPGA Express.

This chapter includes the following sections:

- Using Memory Elements With Lucent Devices
- Mixing HDL and Netlist Input
- Creating a Finite State Machine (FSM)
- Using DesignWare Foundation Components
- Using Encrypted Intellectual Property

The first step in the design flow is to create HDL source files. You can use any text editor to enter VHDL and Verilog HDL source code for your design. FPGA Express source files can be any combination of VHDL, Verilog HDL, EDIF, or XNF netlist files.
This step is usually performed outside of FPGA Express. However, you can use the integrated HDL Editor (described in the section “Debugging Design Source Files” on page 5-6) to edit a design source file once it has been read into a project.

Note:
Using Memory Elements With Lucent Devices

You can synthesis designs with RAM or ROM elements in FPGA Express by using these steps:

To synthesize designs with RAM elements:

1. Generate the RAM netlist using SCUBA, Lucent's Synthesis Compiler for User programmable Arrays. To place the RAM into a VHDL design, choose VHDL output from SCUBA. To place the RAM into a Verilog design, choose Verilog output from SCUBA. SCUBA generates a VHDL entity/architecture pair or Verilog module composed of ORCA library primitives.

2. For VHDL, instantiate the RAM from SCUBA in your design, create a component declaration for the RAM, and place the declaration in the same architecture as the instance. For Verilog, you only need to instantiate the RAM. For an example of a VHDL and Verilog design containing an SSPRAM, see Example 4-2.

3. Read the HDL files for your design and the HDL netlists generated by SCUBA into FPGA Express.

4. Synthesize the design.
To synthesize designs with ROM elements:

1. Prepare a ROM memfile for the SCUBA program. This file contains the values for each location in the ROM. Example 4-1 shows an example of a memfile for a 16x4 ROM. It is an ASCII file with the format addr: value value value.

Example 4-1 Memfile for a 16x4 ROM

00: 0 1 2 3
04: 4 5 6 7
08: 8 9 A B
0C: C D E F

For detailed information, see the *Lucent SCUBA Reference Manual*.

2. Generate the ROM netlist using SCUBA. Choose BusA0 as the bus style and EDIF as the output format. SCUBA generates an EDIF netlist of ORCA primitives.

3. For VHDL, instantiate the ROM from SCUBA in your design, create a component declaration for the ROM, and place the declaration in the same architecture as the instance. For Verilog, you only need to instantiate the ROM. For an example of a VHDL and Verilog design containing a 16x4 ROM, see Example 4-3.

4. Read the HDL files for your design and the EDIF netlists generated by SCUBA into FPGA Express.

5. Synthesize the design.

Example 4-2 RAM Example of a SCUBA Generated VHDL Netlist

```vhdl
-- begin netlist
library IEEE;
use IEEE.std_logic_1164.all;

entity my_sspram is

Creating Design Source Files: Using Memory Elements With Lucent Devices
4-4```
port (addr: in std_logic_vector(3 downto 0); datain: in std_logic_vector(3 downto 0);
  clk: in std_logic; wpe: in std_logic; wren: in std_logic;
  dataout: out std_logic_vector(3 downto 0));
end my_sspram;

architecture Structure of my_sspram is

-- internal signal declarations
signal wpe_inv: std_logic;

-- local component declarations
component RCF16X4Z
  port (AD0: in std_logic; AD1: in std_logic; AD2: in std_logic;
    AD3: in std_logic; DI0: in std_logic; DI1: in std_logic;
    DI2: in std_logic; DI3: in std_logic; CK: in std_logic;
    WREN: in std_logic; WPE: in std_logic; TRI: in std_logic;
    DO0: out std_logic; DO1: out std_logic; DO2: out std_logic;
    DO3: out std_logic);
end component;
component INV
  port (A: in std_logic; Z: out std_logic);
end component;

begin
  -- component instantiation statements
  INV_0: INV
    port map (A=>wpe, Z=>wpe_inv);

  mem_0: RCF16X4Z
    port map (AD0=>addr(0), AD1=>addr(1), AD2=>addr(2), AD3=>addr(3),
      DI0=>datain(0), DI1=>datain(1), DI2=>datain(2), DI3=>datain(3),
      CK=>clk, WREN=>wren, WPE=>wpe, TRI=>wpe_inv, DO0=>dataout(0),
      DO1=>dataout(1), DO2=>dataout(2), DO3=>dataout(3));
end Structure;

-- end netlist

Use the port information in this netlist to generate the instance and
component declaration for the instantiating architecture:

architecture struct of mem_top is

  component my_sspram

Creating Design Source Files: Using Memory Elements With Lucent Devices
4-5
Example 4-3  RAM Example of a SCUBA Generated Verilog Netlist

module my_sspram (addr, datain, clk, wpe, wren, dataout);
    input [3:0] addr;
    input [3:0] datain;
    input clk;
    input wpe;
    input wren;
    output [3:0] dataout;

    INV INV_0 (.A(wpe), .Z(wpe_inv));
    RCF16X4Z mem_0 (.AD0(addr[0]), .AD1(addr[1]),
        .AD2(addr[2]), .AD3(addr[3]), .DI0(datain[0]),
        .DI1(datain[1]), .DI2(datain[2]), .DI3(datain[3]),
        .CK(clk), .WREN(wren), .WPE(wpe), .TRI(wpe_inv),
        .DO0(dataout[0]), .DO1(dataout[1]),
        .DO2(dataout[2]), .DO3(dataout[3]));
endmodule

Use the port information in this netlist to generate the instance for the instantiating module:

my_sspram u0 (addr, datain, clk, wpe, wren, dataout);

Example 4-4  ROM Example of a SCUBA Generated EDIF Netlist

architecture struct of top is

    component my_rom
        port(addr3, addr2, addr1, addr0: in std_logic;

Creating Design Source Files: Using Memory Elements With Lucent Devices
4-6
dataout3, dataout2, dataout1, dataout0: out std_logic);
end component;

begin -- struct
rom_i : my_rom
  port map(addr3 => a(3),
           addr2 => a(2),
           addr1 => a(1),
           addr0 => a(0),
           dataout3 => z(3),
           dataout2 => z(2),
           dataout1 => z(1),
           dataout0 => z(0));

-- rest of architecture

Use the port information in the EDIF netlist to generate the instance and component declaration for the instantiating architecture.

For Verilog, instantiate the ROM entity from SCUBA in your design. For example,

my_rom rom_i(
    .addr0(a[0]),
    .addr1(a[1]),
    .addr2(a[2]),
    .addr3(a[3]),
    .dataout0(z[0]),
    .dataout1(z[1]),
    .dataout2(z[2]),
    .dataout3(z[3]));

---

**Mixing HDL and Netlist Input**

You can mix HDL and netlist inputs by using these steps:

1. Create the appropriate netlist format.
If your design contains schematics, you must export them to EDIF format in your schematic editor.

2. Instantiate the netlist and HDL modules.

Case 1: HDL module instantiating netlist modules

The netlist modules must be instantiated in the HDL code as black boxes.

- In VHDL: Declare the corresponding components and instantiate them. Do not declare any architecture block for these modules—the description is derived from the netlist.

- In Verilog: Instantiate the corresponding modules. Also include empty modules to provide port direction information.

Case 2: Netlist module instantiating HDL modules

The HDL modules must appear in the netlist as black boxes. Their functionality will be derived from synthesis.

3. Synthesize your design.

a. Add all the netlist and HDL files to your project.

b. Set XNF bus style for the input netlists if your target technology is Xilinx. You must specify how the buses of the design are expanded into individual signals because XNF has a wide variety of bus styles.

If the FPGA tool cannot determine which individual signals in the XNF netlist to map to the bus signals defined in the HDL code, you might get a Failed to Link error message.

To set the XNF bus style for the input netlists choose Synthesis > Options > Project > XNF Bus Style and set the switch to the bus format used in the XNF files. For example, if the buses
Creating Design Source Files: Creating a Finite State Machine (FSM)

have the format name<number> in the XNF netlist, set the
XNF Bus Style to %s<%d>. Note that this feature is provided
for backward compatibility.

c. Select the top-level design and create an implementation.

Creating a Finite State Machine (FSM)

This section describes how to create an FSM and contains a VHDL
template and a Verilog template that you can use to create any FSM.
These templates produce optimal synthesis results in FPGA
Express.

Creating an FSM Using a VHDL Template

In VHDL, you can enumerate states symbolically, or you can assign
values to states by using the ENUM.Encoding attribute to control
the state encoding.

FPGA Express supports automatic FSM encoding for enumerated
types in VHDL. To use automatic FSM encoding,

1. Use the VHDL template and omit its ENUM.Encoding
   attribute.

2. Open the Project dialog box (choose Synthesis > Options).

3. Click the Project tab.

4. Click One Hot, Binary, or Zero One Hot encoding. The default is
   One Hot.

The template is a simple counter. The enumerated type for states is
in bold:
library IEEE;
use IEEE.std_logic_1164.all;

entity shift_enum is
  port (CLK,RST : in bit;
        O : out std_logic_vector (2 downto 0));
end shift_enum;

architecture beh of shift_enum is

  type state_type is (S0, S1, S2);
  -- Do not use the following 2 lines for automatic FSM extraction:
  attribute ENUM_ENCODING: STRING;
  attribute ENUM_ENCODING of state_type: type is "001 010 100";

  signal machine : state_type;

  begin
    process (CLK,RST)
    begin
      if RST= '1' then
        machine <= S0;
      elsif CLK= '1' and CLK'event then
        case machine is
          when S0 => machine <= S1;
          when S1 => machine <= S2;
          when S2 => machine <= S0;
        end case;
      end if;
    end process;

    with machine select
    0 <= "001" when S0,
        "010" when S1,
        "100" when S2;
  end;

  You do not need to use a when others statement in this template.

Creating Design Source Files: Creating a Finite State Machine (FSM)
4-10
Following are some tips for describing an FSM in VHDL:

- To ensure that FPGA Express can extract and re-encode the state vector, use an enumerated type for states. For example:

  ```vhdl
  TYPE state_type IS (start, sa, sb, sc, sd, se, sf, sg);
  ```

  Also, from the Project tab, you must select one of the following default encoding synthesis options: Binary (for a smaller but slower implementation) or One Hot (for a faster but larger implementation). To select these options, choose Synthesis > Options and click the Project tab.

- To describe the logic of the next state and outputs, use a CASE statement. For example:

  ```vhdl
  process (clk, rst)
  begin
    IF rst='1' THEN
      current_state <= start;
      outputs <= x;
    ELSIF clk = '1' AND clk'event THEN
      CASE current_state IS
        WHEN start =>
          <Describe next state and outputs>
        WHEN sa =>
          <Describe next state and outputs>
          .
          .
          .
      END CASE;
    END IF;
  END PROCESS;
  ```

- For one-hot encoding, if recovery is not needed, omit the when others statement from the VHDL source code.

  The statement
when others =>
next_state <= start;

in the next_state logic description covers all states that you do not specify (including invalid states). For example, any state with more than one bit of the state vector at logic one is invalid in a one-hot implementation.

If a transition to an invalid state occurs, the FSM reverts to state start on the following clock cycle. The FPGA tool generates next_state logic to ensure that the implementation exactly matches the VHDL description and that the FSM can recover from invalid state transitions.

If you do not need recovery from invalid state transitions, you can generate a smaller and faster implementation of the one-hot FSM by removing the when others statement from the VHDL source code. Or, if you do not remove the when others statement, select the fastest & smallest FSM Synthesis option (choose Synthesis > Options and click the Project tab).

If you do not specify enumerated states in the next state logic description, you must either specify or remove them from the enumerated type definition. This eliminates the selection logic for all of the invalid states from the when others statement, which makes a one-hot FSM smaller and faster.

---

**Creating an FSM Using a Verilog Template**

Although Verilog does not support enumerated types, you can use this Verilog template to design an FSM in which you fully control the state encoding.
To use this template:

- You must use the parameter statement to define state values.
- You can use a CASE statement and the Synopsys directive
  \//synopsys parallel_case full_case to describe the state
  machine. The parallel_case directive indicates that all states are
  mutually exclusive. The full_case directive indicates that all
  possible states are specified.

The template defines the same simple counter as in the VHDL
template. The parameter statement for state values is in **bold**:

```vhdl
module shift (clk, rst, out);
input clk, rst;
output [2:0] out;
parameter [2:0]
    S0 = 3'd1,
    S1 = 3'd2,
    S2 = 3'd4;

reg [2:0]
    state, next_state;

always @ (state)
begin
    case (state) // synopsys parallel_case full_case
        S0: next_state = S1;
        S1: next_state = S2;
        S2: next_state = S0;
    endcase
end

always @ (posedge clk or posedge rst)
begin
    if (rst) state <= S0;
    else state <= next_state;
end

assign out = state;
```
You can implement FSMs as either safest (for recovery from invalid state transitions) or fastest and smallest (when recovery is not needed).

**Using Encrypted Intellectual Property**

FPGA Express lets you use encrypted intellectual property (IP). This feature lets you focus on designing at the system level (rather than on redesigning common functions).

You can instantiate encrypted Verilog or VHDL designs. When FPGA Express encounters encrypted IP, it decrypts the IP before synthesis. FPGA Express uses architecture-specific optimization algorithms.

To ensure maximum protection of IP functionality, if it finds any encrypted instantiated IP module, FPGA Express encrypts the entire design netlist.

Netlist encryption disables the following features of FPGA Express for the entire chip:

- View Schematic command for elaborated and optimized designs
- View Results command for optimized designs
- Verilog and VHDL simulation output for the Export Netlist command
- Project Report command
- Chip Report command
- File Report command (for the encrypted file only)
- Display of hierarchy of elaborated or optimized design
- Display of modules in a file (for the encrypted file only)
- The Clocks, Paths, Modules, and Registers constraint tables
- Export of .db files

Also, netlist encryption disables FPGA scripting tool (FST) commands that correspond to these features.

For more information about IP encryption and disabled FST commands, see the FPGA Express release notes (the readme.htm file), which are in your installation directory.
Setting Up Projects

This chapter describes how to set up projects in FPGA Express. It contains the following sections:

- **Creating a Project**
- **Specifying Design Source Files**
- **Debugging Design Source Files**

An FPGA Express project is a working directory that holds a project file and subdirectories for internal and intermediate files. The key components of a project are the design source files that describe the FPGA design.

FPGA Express supports design source files in the following formats: VHDL, Verilog, EDIF, and XNF netlists. Netlist files are usually from schematic capture systems or preoptimized macro libraries.
Creating a Project

After you create the design source files, you create a project. FPGA Express makes a new folder (a directory) to store the information for the new project.

To create a project in FPGA Express,

1. Launch FPGA Express.
   To do so, choose Start > Programs > Synopsys > FPGA Express version > FPGA Express.

   FPGA Express opens the main window, which includes the toolbar, the tip bar, the Design Sources window, the Chips window, and the Output window.

   (If the DesignWizard window appears, click Cancel. The DesignWizard is another way to create a project. For information about the DesignWizard, see “DesignWizard Flow” on page 2-2.)

2. Create a new project.
   To do so, do one of the following:
   - Click the New Project button in the toolbar.
   - Choose New Project from the File menu.

   In the main window, FPGA Express displays the Create New Project dialog box.
The dialog box includes a default location and name for the project. You can change the location, name, or both. If you want to do this, use the drop-down list in the Save In field to navigate through your directory tree or click the Create New Folder icon to create a new directory.

3. In the Name field, type a name for your project.

FPGA Express will store the project files in a directory that has the same name as your project.

4. Click the Create button.

After the tool creates the project, the Add Sources dialog box opens.

---

**Specifying Design Source Files**

After you create the project, you add the design source files to it. When you add files, FPGA Express analyzes them automatically to detect errors. FPGA Express analyzes files using the Synopsys industry-standard HDL language support.

When you create a project, a default library named WORK appears in the project folder. You add the design source files to the WORK library. You can add any combination of VHDL, Verilog, EDIF, and XNF netlist files.

VHDL environments sometimes need multiple libraries. You can add a library at any time by right-clicking in the Design Sources window and choosing New Library. Then, you add design source files to that library.
The order in which you add files can be important. For example, if a package defined in file1 uses another package defined in file2, you must add file2 before adding file1. This order of addition makes the tool analyze file2 before file1. However, you can change the order later (for more information, see the online Help).

If the design source files contain errors, the Output window helps you find and correct problems. You can use the integrated text editor (the HDL Editor) to debug the files. For explanations of error messages, see the online Help.

FPGA Express does not copy design source files to another directory. When you add files, the tool analyzes them in their current location. If you changed the files (a ? icon beside a file name indicates a change), you must reanalyze them by clicking the Update button on the toolbar or right-clicking in the Design Sources window and Update File.

To add design source files,

1. Open the Add Sources dialog box.

   To do this, do one of the following:

   - Click on the toolbar.

   - Choose Synthesis > Add Source Files.

   - Right-click WORK in the Design Sources window and choose Add Sources in WORK.

   The Add Sources dialog box appears.

2. In the Add Sources dialog box, click or shift-click to highlight the design source file or files.
3. Click Open.

On a Windows PC, you can also add files to the project by dragging them into the Design Sources window.

FPGA Express displays the project window and extends the menu bar. The project window has two windows—the Design Sources and Chips window (see Figure 5-1).

*Figure 5-1 The Project Window After You Add Design Source Files*

The Design Sources window displays the name, location, and status of each of the design source files. The Chips window displays information (such as name and device type) for individual design implementations. The project window’s title bar displays the name of the project.
In the Design Sources window, the icon to the left of each file name indicates the results of the analysis. For example, in the figure, the red cross indicates that the counter file has at least one error. The green check marks next to the other files indicate that the files have no errors or warnings.

The error in the design source file is reflected up the hierarchy. Therefore, the library and the project icons are also marked with red crosses. (For a list of analysis status icons and their explanations, see the online Help.)

The Output window (which is at the bottom of the screen) displays error messages about the selected design source file.

---

**Debugging Design Source Files**

You use the Errors, Warnings, and Messages tabs of the Output window to find and correct errors and warnings in design source files. The Output window shows the file name, line number, and type of each error or warning. After viewing the messages, you can use the HDL Editor to investigate and fix the errors and warnings.

To view and correct errors and warnings,

1. In the Design Sources window, click to highlight the design and read its errors and warnings in the Output window.

2. Open the HDL Editor and correct any errors.

   You can open the HDL Editor in either of the following ways:

   - Right-click in the Design Sources window and choose Edit File.
- Double-click an error or warning message in the Output window.

FPGA Express opens the HDL Editor so that you can edit the file. You can use the HDL Editor Edit menu to navigate in the file. Also, you can use the pop-up menu that appears when you right-click in the HDL Editor window to navigate.

You can use an editor other than the HDL Editor as the default design source file editor. To do this, choose Synthesis > Options, click the General tab, and deselect the Use FPGA Compiler II internal source editor check box. If you deselect the check box, choosing Edit File opens the file in the editor associated with the file type.

Figure 5-2 shows the HDL Editor window and its pop-up menu.

**Figure 5-2 HDL Editor Window**

To display this menu, right-click in the HDL Editor window.

In line 20 of Figure 5-2, the HDL Editor indicates that a misspelling in the `if` statement caused the error in the design named counter.
3. Save the file.

   When a file is out of date, the file icon contains a question mark.

4. Update the file.

   To update the file, do one of the following:

   - Go to the Design Sources window and click on the toolbar.
   - Right-click in the HDL Editor window and choose Analyze File (before you close the HDL Editor window).

Depending on the extent of your changes, FPGA Express updates the file, the library, or the project. FPGA Express reanalyzes only the files that you change.

After adding, analyzing, and debugging source files, you can elaborate the design.
Elaborating Designs

This chapter describes how to elaborate designs in the FPGA tool. It contains the following sections:

- Elaborating a Design
- Using Block-Level Incremental Synthesis (BLIS)
Elaborating a Design

Elaboration creates a unique, unoptimized implementation from your design source files. The FPGA tool translates the HDL design description to a logical gate-level description that uses technology-independent primitives.

To elaborate a design, you identify the design that you want to use as the top level, and you set implementation options (such as target device, clock speed, and optimization goal).

Identifying the Top-Level Design

To start building an implementation from analyzed source files, you identify the top-level entity (for VHDL), module (for Verilog), or schematic netlist as the top-level design. The FPGA tool uses the top-level design that you identify to build design hierarchy and interconnections.

Identify the top-level design in any of the following ways:

- From the drop-down list of top-level designs (entities, modules, and netlists) in the toolbar, select the top-level design that you want to use (see Figure 6-1).

Figure 6-1 Drop-Down List of Top-Level Designs in the Toolbar
- In the Design Sources window, double-click the design source file that you want to use. This expands the file and displays an icon for the file. Then, right-click the icon and choose Create Implementation.

- Click to highlight the icon of the top-level design and click the Create Implementation button on the toolbar.

After you identify the top-level design (in any of the three ways above), the Create Implementation dialog box appears (see Figure 6-2).

**Figure 6-2  Create Implementation Dialog Box**

![Create Implementation Dialog Box Image]

---

**Creating the Implementation**

When you create an implementation, the tool elaborates logic for each source file. It determines the complete hierarchical structure and topology of the design (including multiple-level links and references between subdesigns). The tool uses this information to produce an intermediate, unoptimized implementation.
To create the implementation,

1. In the Implementation Name field, enter an implementation name.
   
   If you do not enter a name, the FPGA tool uses the name of the top-level design to create a unique implementation name.

2. Select the target device and speed grade for the design.

3. Set the default clock frequency.
   
   All clocks in the design use this default frequency. (After you elaborate the design, you can use the design constraint tables to change default clock frequencies later.)

   Over constraining a design can degrade the place and route results. Therefore, you should specify only what you need.

4. Choose the optimization goal and optimization effort.
   
   You can optimize for speed or area and high, fast, or low CPU effort. For more information, see the online Help.

5. Select or deselect the Do Not Insert I/O Pads option.
   
   This option determines whether the tool inserts I/O pads during optimization. Use this option for implementations that are modules in larger designs. For more information, see the online Help.

6. Select or deselect the Skip Constraint Entry option.
   
   This option enables or disables constraint specification for the design. For more information, see the online Help.

7. Select or deselect the Preserve Hierarchy option.
This option controls whether the tool preserves or eliminates the module’s boundary during optimization. For more information, see the online Help.

8. Click OK.

The design is elaborated. An implementation icon, implementation name, target device, and speed grade appear in the Chips window. The icon indicates the implementation status (see Figure 6-3).

**Figure 6-3 Creating the Design Implementation**

9. View the Output window to investigate any errors and warnings.
This chapter describes how to set constraints, attributes, and controls in the FPGA tool. It includes the following sections:

- Importing Constraints
- Entering Constraints, Attributes, and Controls
- Editing Table Cells
- Setting Multicycle Timing Constraints

Before you start to optimize a design for a target device, you can set performance constraints, attributes, and optimization controls. Design constraints guide the FPGA tool with specific optimization requirements.
Using the graphical user interface, you can enter constraints for your design in editable tables. The constraints contain performance requirements and optimization options for architecture-specific optimization engines.

Constraint information can be exported to ASCII files for easy editing. For more information, see “Exporting an ASCII Constraint File” on page 12-10.

The FPGA tool separates constraint entries into logically related groups (for example, clocks, paths, ports, and modules). The tool automatically extracts design-specific information, such as clock names, port names, and design hierarchy, from the design and displays it in tables. You enter performance constraints, attributes, and optimization options directly into the tables.

Each set of constraint tables and dialog boxes is specific to a particular target FPGA architecture.

Specifying constraints is optional, but highly recommended. Entering your requirements in the constraint tables can significantly improve place and route results.

For example, if you enter constraints for an output port with restrictive speed requirements, it is easier for the place and route tool to fulfill those requirements. If a design is very large and has many hierarchical levels, you can improve place and route results by entering hierarchy constraints. Note, however, that overconstraining a design can adversely affect place and route results. Therefore, specify only what you really need.
**Importing Constraints**

You can import an ASCII file of constraint commands for the current chip by using menu commands or the scripting tool. For more information, see “Importing an ASCII Constraint File” on page 12-11 for instructions.

---

**Entering Constraints, Attributes, and Controls**

To enter design constraints, attributes, and controls,

1. Right-click the elaborated implementation and choose Edit Constraints to open the Constraints window.

   **Figure 7-1** shows how the constraints and synthesis controls (Altera options) are logically grouped by function into separate Clocks, Paths, Ports, and Modules tabs. An additional tab, called Registers, is available for selected FPGA technologies.

   **Figure 7-1  Constraint Tables**
2. Click the tabs to display tables for the constraint groups.
   
   For details about specific constraints, attributes, and optimization options, see the online Help.

3. Explore the tables for the implementation.
   
   The contents of the tables reflect the architecture you chose earlier. Note that the Clocks and Paths constraint tables are preloaded with the clock frequency (and corresponding period) that you entered for the target clock frequency.

4. After entering any constraints, attributes, and option information, close the Constraint window. This action automatically saves any changes you’ve made in the tables.

---

**Editing Table Cells**

To edit table cells,

1. Click in the table cell.
   
   A down arrow appears.

2. Click the down arrow.
   
   A menu or editable field appears.

3. Choose a menu item or enter text as appropriate.
Setting Multicycle Timing Constraints

You can use the FPGA tool to generate timing groups and path groups for logic that uses clock enable signals. This feature is useful when parts of a design run at a slower speed than the rest (with the enable signal controlling the slower flip-flops).

Using enable signals with a fundamental clock eliminates clock skew, which additional clock signals can introduce. You can set multicycle timing constraints for specified paths, which makes the slower logic easier to place and route. You can apply multicycle timing constraints to subpath groups that you create.

Figure 7-2 is an example in which a multicycle timing constraint is appropriate. The FPGA contains high-speed interface logic that must run at 40 MHz and low-speed interface and core logic that must run at 10 MHz. The FPGA has a 40-MHz system clock and uses the clock to generate a 10-MHz enable signal for internal distribution.

Figure 7-2  Shift Register With Multicycle Timing Constraint Assignment
The following example shows how the 10-MHz enable signal might align with the system clock when the rising edge of the 40-MHz system clock is the active edge. The 40-MHz clock is distributed to the clock input of each FPGA flip-flop, and the enable signal is distributed to each FPGA flip-flop clock enable input. The primary clock period is 25 ns, but the 10-MHz enabled logic must satisfy a period of only 100 ns:

```
40 MHz clock     _|_|_|_|_|_|_|_|_|_|_|_|_|_|_ 10 MHz enable      ______|--|____________|--|___________
```

The simple shift register circuit in the logic diagram in Figure 7-2 shows how the multicycle timing constraints are assigned.

Register reg1 is a 4-bit, serial-input, parallel-output register. Register reg2 is a holding register that is loaded with the clock enable signal ena. The paths from the output of reg1 to the input of reg2 (net q) are multicycle paths, because the data bits have four clock cycles to reach their destinations. The register-to-register timing constraint is 25 ns (1/40 MHz), but the multicycle timing constraint is 100 ns (4 x 25 ns).

To create a subpath group of the register-to-register paths,

1. Open the Paths constraint table.
2. Right-click the register-to-register path groups.
3. Choose New Sub path.

The Create/Edit Timing Sub Path window appears.

4. Use the Create/Edit Timing Sub Path window to construct your own path group by choosing specific startpoints and endpoints.

Figure 7-3 shows the Create/Edit Timing Sub Path window.
The new path group is called a subpath, because it is a subset of another path group; in this case, the register-to-register paths in the design.

In the example in Figure 7-4, the outputs of reg1 are the startpoints, and the inputs of reg2 are the endpoints for the subpath. A delay of 100 ns is assigned to the subpath.
For more information about using the Create/Edit Timing Sub Path window, see the online Help.

After you create a subpath and apply the multicycle timing constraint, the subpath appears in the Paths constraint table, as shown in Figure 7-5.
You can include an enabled flip-flop in two path groups—those that include clock-to-clock paths and those that include clock-to-enabled-clock paths. This implies that there are two overlapping constraints. The constraint for clock-to-clock timing (25 ns in this case) conflicts with the constraint for clock-to-enabled-clock timing, which is 100 ns.

Most vendor place and route tools assign different priorities to these two constraints by placing a higher priority on the more specific one. Because the subpath constraint is more specific than the clock-to-clock constraint, it takes precedence, and the corresponding paths can be optimized for the slower speed.
Setting Constraints, Attributes, and Controls: Setting Multicycle Timing Constraints
This chapter describes how to optimize design implementations in FPGA Express. It includes the following sections:

- Setting General Synthesis Options
- Setting Project Synthesis Options
- Updating and Force-Updating a Project After Incremental Changes

After entering constraint, attribute, and option information, you are ready to optimize the design implementation.

In this step, you optimize the design implementation elaborated earlier, guided by the constraints and controls you entered in the constraint tables.
To optimize an elaborated implementation,

1. Click the elaborated implementation in the Chips window to select it. Its name is displayed on the top-level design field of the toolbar.

2. Right-click the implementation and choose Optimize Chip, or click on the toolbar.

A new optimized implementation icon appears beneath the original implementation.

When you optimize a design implementation, the actual timing of your design against your requirements is analyzed. After optimization, the design implementation tables display the constraints you specified with your design results so that you can compare them.
Setting General Synthesis Options

You can configure the appearance and behavior of FPGA Compiler II by choosing Synthesis > Options and clicking the General tab. The options that you select apply to all subsequent projects.

Figure 8-1  General Options
From the General dialog box you can select any of the following options:

**Show full pathnames of source files**

By default the names shown for a design source file in the Project window is its file name, without its pathname. If you have large projects with sources from several directories, it can be useful to see the full path names to prevent ambiguity. When you check this box, FPGA Compiler II displays the full path for each file.

**Warn before deleting files and chips**

When you select this option (the default), the FPGA tool prompts before removing any source files or implementations from the project.

**Show Add Files dialog box after creating project**

When you select this option (the default), the FPGA tool automatically prompts for design source files when you create a new project.

For Windows users: You can also drag and drop design source files to add them to the project.

**Using the Internal Editor**

The FPGA tool has an integrated, built-in HDL Editor that you can use to edit design source files. You can find next and previous errors and analyze a file without leaving the editor. By default, this option is selected.

If you have an external editor with which you are more familiar or which provides other useful capabilities, clear this box, and ensure that your editor is associated with the appropriate file extensions (using the standard capabilities integrated with the Windows File Manager or Explorer programs).
Ask before creating directory

Selecting this option prompts the user before directories are created.

Default clock frequency when creating new chip

When you create a new implementation, a default frequency for the primary clock of the design is available. Use this field to set the default as appropriate for your subsequent project.

---

**Setting Project Synthesis Options**

You can configure the appearance and behavior of the FPGA tool by choosing Synthesis > Options and clicking the Projects tab. The options that you select apply to all subsequent projects.
From the Project dialog box, you can select any of the following options:

**FSM synthesis: Default encoding (takes effect on file analysis)**

These buttons control the encoding for finite state machines (FSM) in VHDL designs. Choices include One Hot (default), Binary, and Zero One Hot. See FSM State Assignment.

The shell variable for this feature is `proj_fsm_coding_style`. 
FSM synthesis: Interpretation of VHDL ‘when others’

These buttons control implementation of the ‘when others’ statement in the VHDL case statement describing the state logic. Choices include fastest & smallest (default) or safest.

The “fastest and smallest” selection provides the best implementation when illegal state transitions can be ignored. If recovery from illegal state transition is necessary, select safest. To ensure the fastest and smallest FSM implementation, the ‘when others’ statement from your FSM descriptions is ignored.

The variable for this feature is
proj_fsm_optimize_case_default.

Verilog synthesis (takes effect on file analysis)

The tool includes a new HDL Compiler for Verilog. One of the features that this compiler offers is the ability to implement “fastest and smallest” as well as “safest” FSMs in Verilog the same way it does for VHDL designs.

To enable the new HDL Compiler, select New HDL Compiler – Presto for Verilog HDL Compiler Option. Checking the New HDL Compiler box makes the “fastest and smallest” as well as “safest” switches for FSM synthesis available to Verilog FSMs.

The variable for enabling the new HDL Compiler for Verilog is proj_compiler_type. See the man page for syntax and usage.

Note:

You must Force Update your design source files for the change to the new HDL Compiler to take effect.

Default Export Timing constraints option to YES

This check box sets the default for the Export Timing Specifications option when you export a netlist.
The variable for this feature is
proj_export_timing_constraint.

Enable Verilog Preprocessor

Verilog Preprocessor constructs such as 'ifdef, 'else, 'endif are disabled by default. To enable the Verilog Preprocessor for subsequent HDL analyses of Verilog files, open the Project Options dialog box and check the Enable Verilog Preprocessor box.

The variable for this feature is proj_enable_vpp.

Note:
You must reanalyze any previously analyzed files. Otherwise changes do not affect the synthesized RTL.

Export Directory

You can specify the default directory in which netlists are stored during Export Netlist.

The variable for this feature is proj_export_directory.

Export Directory

Specify whether the tool saves these settings as user defaults for new projects.

This check box controls saving the dialog-box settings for other new projects.
Updating and Force-Updating a Project After Incremental Changes

If your changes are not evident to FPGA Express (for example, if the changes are in a dependent file), right-click in the Design Sources window and choose Force Update File (Library or Project).
This chapter describes how to use the schematic viewer to view schematics and timing results. It contains the following sections:

- Viewing a Schematic of an RTL (Generic) Design
- Viewing a Schematic of an Optimized (Mapped) Design

You can view and analyze your design graphically by using the integrated schematic viewer. You can view an RTL version of the design, as shown in Figure 9-1, or an optimized (mapped) version of the design, as shown in Figure 9-2.
Figure 9-1  RTL Version of a Design in the Schematic Viewer
Figure 9-2  Optimized (Mapped) Version of a Design in the Schematic Viewer
Viewing a Schematic of an RTL (Generic) Design

You can use the schematic viewer to examine a variety of information about an RTL (generic) design. This information includes design hierarchy and the contents of a block.

To view the schematic of an RTL design,

1. In the Chips window, right-click the elaborated implementation.
2. From the pop-up menu, choose View Schematic.
   
   The schematic viewer appears. The viewer windows arrange themselves to increase viewable area.

3. Maximize the main window.
4. Navigate within the schematic in any of the following ways:
   
   - To zoom the schematic, click the Zoom In, Zoom Out, Zoom In Tool, or Zoom Full-Fit buttons on the toolbar.
   - To zoom into a specific location, click the Zoom In Tool button on the toolbar and drag the pointer over the location.
   - To view the contents of a block, double-click the block.
   - To return to the next level up, right-double-click anywhere in the schematic.
   - To navigate around the hierarchy by using the Chips window, double-click a level of hierarchy to expand it or select one of the blocks to view that level.
Viewing a Schematic of an Optimized (Mapped) Design

You can use the schematic viewer to navigate within and zoom in to and out of an optimized (mapped) design. The schematic viewer lets you view the project files, the TimeTracker window (see “Viewing Timing Results Graphically Using TimeTracker” on page 10-5), and the schematic window at the same time.

To view the schematic of an optimized design,

1. In the Chips window, right-click the optimized implementation.

2. From the pop-up menu, choose View Schematic.

   The windows arrange themselves to display the project window on the left and the TimeTracker and schematic windows on the right.

3. You can navigate within the schematic in any of the following ways:
   - To zoom the schematic, click the Zoom In or Zoom Out Tool, or Zoom Full-Fit buttons on the toolbar.
   - To zoom into a specific location, click the Zoom In Tool button on the toolbar and drag the pointer over the location.
Viewing Schematics: Viewing a Schematic of an Optimized (Mapped) Design

9-6
Viewing and Debugging Timing Results

This chapter describes how to view and debug timing results. It includes the following sections:

- Analyzing Timing
- Viewing Timing Results Graphically Using TimeTracker
Analyzing Timing

After you create an elaborated implementation, you can determine circuit performance by checking the results of optimization and analyzing timing information. The post-synthesis timing data is displayed in the same formats as the tables you used to enter constraints.

To view the results of optimization,

1. Right click an optimized implementation and choose View Results.

2. Check the Clocks constraint table to see the maximum clock frequencies that were estimated for each of the clocks in the design. Clock frequency violations appear in red. Figure 10-1 shows the Clocks constraint table after optimization.

   **Figure 10-1  Optimization Results in the Clocks Constraint Table**

   ![Clocks Constraint Table](image)

3. Check the Paths constraint table for more information about timing violations:
   - Select a path group to see a list of paths in that group.
   - Select a path from the list to see the details of path composition, cumulative delays, and fanout.
- See the topic-based online Help for a description of timing analysis in FPGA Express.

Figure 10-2 shows the Paths constraint table after optimization.

Note:
All pins of primitive cells on the timing path are displayed. Therefore two rows on the right window correspond to a single cell (source and load).

**Figure 10-2 Optimization Results in the Paths Constraint Table**

4. Check the Ports constraint table for information about input and output delays.

Figure 10-3 shows the Ports constraint table. The results include the slack for input arrival time and output delay for each port.
5. Check the Modules constraint table for information about the device resources used. Double-click the items in the Area column for details about cell count.

Figure 10-4 shows the Modules constraint table after optimization.
Viewing Timing Results Graphically Using TimeTracker

You can use TimeTracker to examine a variety of timing results for a specific path in a design. These results include critical path, cell type, delay, fanin and fanout, and slack values.

To view timing results for a path in the schematic viewer,

1. Click the Paths tab.

   The Paths constraint table appears.
2. In the path groups window (the upper-left window), click to choose the row for the path group.

   The paths constrained by the path group appear in the paths area (the lower-left window). For any paths that fail timing, the timing values appear in red in the TimeTracker window. In the schematic, the paths are highlighted.

3. From the group of paths, select the path for which you want to view timing results.

   The right side of the TimeTracker window displays the cells in this path. The instance name, cell type, delay, and fanout for each cell is displayed.

   In the schematic window, the critical path appears in red.

4. View the information about the path in any of the following ways:

   - To view the cell name, cell type, pin numbers, delay, fanout, and slack values, choose the path and hold the pointer over the cell. A pop-up window displays this information.

   - To find the cell to which a pin is attached, choose the pin from the pin list. The schematic window highlights the cell in yellow.

   - To display the fanin and fanout logic cones, make sure that the cell is selected and highlighted in yellow, then click the Fan In and Fan Out buttons on the toolbar.

   - To move along the path, click the Previous Pin and Next Pin buttons on the toolbar or click each pin in the path in TimeTracker.
This chapter describes how to generate netlists and reports and how to use Quartus II with FPGA Express. It includes the following sections:

- Generating a Netlist
- Generating a Report
- Generating a Project Database
- Using Quartus II

When the design implementation is optimized, you can generate two kinds of netlist files:

- EDIF or XNF files formatted for place and route by FPGA vendor tools
- VHDL or Verilog files for functional simulation

You can generate a report file to review and document the project.

You can also export the project database in .db format to enable integration with other Synopsys tools.

---

**Generating a Netlist**

FPGA Express generates EDIF or XNF netlists that place and route tools from FPGA vendors or Quartus II can process directly. The tool can also generate VHDL and Verilog netlists for simulation.

To generate netlist files,

1. Either select the optimized implementation and click ![icon] on the toolbar, or right-click the implementation and choose Export Netlist. The Export Netlist dialog box appears (see Figure 11-1).

![Figure 11-1 Export Netlist Dialog Box](image)

2. Choose an export directory for the netlist files.
3. To change directories, either type the new directory name or click Browse.

4. Specify whether to export timing constraints with the netlist, using the Export Timing Specifications check box.

5. Select an output format for your netlist:
   - To export only the netlist for place and route, choose NONE in the Output Format list box.
   - To also export a netlist for simulation, choose Verilog or VHDL.

6. Click OK.

   **Caution!**
   Many files might be exported from a single design. To avoid overwriting your source files, always export netlists into a separate directory.

---

**Generating a Report**

You can generate the following reports:

- **Library report**
  A library report shows errors, warnings, and messages for each design source file in the library.

- **File report**
  A file report shows errors, warnings, and messages for the selected design source file.

- **Chip report**
A chip report shows implementation settings such as target device, synthesis goal, optimization effort, clock frequency, and other timing information.

- Project report
  
  A project report contains all the information in the library, file, and chip reports.

To generate a report,

1. In the project window, choose the project, library, file, or chip and click on the toolbar.

   Or, right-click the project, library, file, or chip and choose Report from the pop-up menu.

   For more information about opening the project window, see Chapter 5, “Setting Up Projects.”

2. In the dialog box that appears (see Figure 11-2), specify the name and location for the report.
3. Click Save. A text file containing summary information for the whole project, the library, the design file, or the chip is created.

4. Open the report file in a text editor or word processor.

Generating a Project Database

The FPGA tool can generate project databases in .db format. The .db format enables integration with other Synopsys tools.

To generate project database files,

1. Either select the optimized design implementation and click on the toolbar, or right-click the implementation and choose Export Netlist. The Export Netlist dialog box appears (see Figure 11-3).
2. Specify whether to export the Synopsys design database with the netlist, using the Generate Synopsys db Files check box.

Using Quartus II

You can launch Quartus II from within the GUI. All the files that Quartus II requires are automatically generated in the directory specified in the Place and Route dialog box in Figure 11-4.

To launch Quartus II,

1. Either select the optimized implementation and click on the toolbar or right-click the implementation and choose Place and Route Chip. The Place and Route dialog box appears.
2. Choose a place and route directory.

3. To change directories, either type the new directory name or click Browse.

4. Specify whether to export timing specifications with the netlist, using the Timing Specifications check box.
Using Constraint Files

This chapter describes how to use constraint files with FPGA Express. It includes the following sections:

- **About the ASCII Constraint File Format**
- **Exporting an ASCII Constraint File**
- **Importing an ASCII Constraint File**

FPGA Express can export and import constraint files in ASCII format.

When you export a constraint file, you create a file that contains the constraint settings of the current chip. You can edit the ASCII constraint file and reapply the new constraint information to the current chip. When you import a constraint file, the tool sets the constraints on the current chip.
This method of exporting, editing, and importing the constraint file is useful if you use a scripting methodology. However, GUI users can find this method of editing constraints useful as well—for example, in order to avoid keyboard and mouse tasks required by data entry in the GUI constraint tables.

Exporting or importing a constraint file always acts on the current chip. Set the current chip with the `current_chip` command before exporting or importing a constraint file.

Note:
To support current methodologies that require doing so, you can continue to import constraint files in binary format. Exporting binary constraint files is no longer supported.
About the ASCII Constraint File Format

In the ASCII constraint file you can find

- Header information that identifies the chip name and target architecture
- Constraint information and commands in sections that correspond to the tabs in the constraint tables.
- Comments that show valid constraint settings and other information

The ASCII constraint file contains constraint information, comments, and defaults for

- Clocks
- Object groups (as commented information in the export file)
- Paths
- Ports
- Modules
- Registers
- Vendor options

Comments in the file begin with a hash mark (#).
**Header Information**

Header information consists of the chip name and the architecture. The header appears at the beginning of the file. Header information is optional when you are importing an ASCII constraint file.

Following is an example:

```plaintext
chip_name = tutor;
arch_name = APEX20K;
```

**Clock Constraints**

The clock constraints portion of the file contains default information and user settings. The first line defines the default clock period, rise time, and fall time. Subsequent lines contain the clock names followed by the clock period, rise time, and fall time for each clock. If the clock uses a default value, -1 appears.

Following is an example:

```plaintext
######## Clock Constraints ########

clock <default> 20/0/10
clock data_clk1 -1/-1/-1
clock clk2 -1/-1/-1
```
Object Groups

Logically related groups of design objects appear as comments in the ASCII constraint file. In the following example, (I) is the group of input objects, (O) is the group of output objects, and (RC, data_clk1) is the group of objects triggered by the rising clock edge of data_clk1.

Example

```
# Members of Group  (I):
#   /d/vcc
#   /d/data_clk1
#   /d/clk2
#   /d/clrn
#   /d/ena1
#   /d/ena2
#   /d/data[3]
#   /d/data[2]
#   /d/data[1]
#   /d/data[0]

# Members of Group  (O):
#   /d/regout
#   /d/vcc
#   /d/data[3]
#   /d/data[3]
#   /d/data[2]
#   /d/data[1]
#   /d/data[0]
#   /d/done

# Members of Group  (RC,data_clk1):
#   /d/u1/q_reg
```

Path Constraints

The path constraints section sets the required delay between objects or object groups.
In the following example, the first line defines required timing in nanoseconds between all inputs (I) and outputs (O). The second line defines required timing in nanoseconds between all inputs and objects triggered by the rising edge of the data_clk1 clock.

Example

######## Path Constraints ########

path -from (I) -to (O) -delay 33
path -from (I) -to (RC,data_clk1) -delay 34

To define subpaths, you define a name for the subpath and the explicit sources and destinations of the subpath.

In the following example, the first path command defines a subpath called my_from:my_to between five input objects and one output object.

In the second path command, the subpath is called m_f:m_t. The subpath starts at input /d/ena2 and terminates at output /d/regout.

For clarity, the export file prints the -from, -to, and -delay objects value on separate lines.

Example

path my_from:my_to
   -from
      (I:
       /d/data_clk1,
       /d/clk2,
       /d/clk2,
       /d/clrn,
       /d/ena1,
       /d/ena2)
   -to
      (O:
       /d/regout)
   -delay 43;
Using Constraint Files: About the ASCII Constraint File Format

Port Constraints

The port constraints section of the constraint file contains user-defined defaults and port constraints.

Example

######## Default Port Constraints ############

  port <default> use_io_reg "OFF"
# Valid 'use_io_reg' values are:
#   <default>
#   ON
#   OFF

  port <default> slew_rate "FAST"
# Valid 'slew_rate' values are:
#   <default>
#   SLOW
#   FAST

Example

######## Port Constraints ############

# Port clk2, direction = input
port clk2 input_delay 34/(RC,data_clk1);

# Port clrn, direction = input

port clrn input_delay 34/(RC,data_clk1);
port clrn location "data location";

# Port done, direction = output

port done output_delay 20/(RC,data_clk1);

Module Constraints

The module constraints section of the constraint file contains user-defined defaults and module constraints.

Example

############# Default module constraints #############

module <default> hierarchy "preserve"
# Valid module ‘heirarchy’ values are:
#     inherit
#     preserve
#     eliminate

module <default> primitive "preserve";
# Valid module ‘primitive’ values are:
#     inherit
#     preserve
#     optimize

Example

############ Module Constraints ########

# module test - Root design
Register Constraints

The register constraints section displays information from the Registers constraint table. The estimated fanout for registers is shown in comment lines.

Example

# Register Constraints :
#--------------------------------
register <default> max_fanout 3;

    register sdatc_reg<11> max_fanout 5;
# register sdatc_reg<11> estimated fanout = 1
# register sdatb_reg<4> estimated fanout = 1

Vendor Options

The vendor options section contains options specific to the target architectures available in FPGA Express.
Example

lucent_option ignore_unlinked_cells "disable";
xilinx_option ignore_unlinked_cells "enable";
xilinx_option buffer_internal_nets "enable";
actel_option max_fanout 16;
actel_option advanced_optimization "disable";
genral_option use_lpm "enable";

Exporting an ASCII Constraint File

You can export an ASCII file of constraint commands for the current implementation by using menu commands or the command line (scripting tool).

To use GUI commands to export an ASCII constraint file,

1. Right-click the elaborated implementation in the Chips window.
2. Choose Edit Constraints from the pop-up menu.
   
   Steps 1 and 2 set the current implementation and enables the menu items in step 3.
   
   The Export Constraint File dialog box opens.
4. Enter the file name you want and click Save.
   
   When saving the file, the tool appends the .scf ending for an ASCII constraint file.

To use the shell to export an ASCII file of constraint commands for the current implementation,

1. Run the scripting tool.
Using Constraint Files: Importing an ASCII Constraint File

See “Using the Shell” on page 3-4 for instructions.

2. Set the current chip. At the shell prompt, enter

    current_chip my_chip

3. At the shell prompt, enter

    export_constraint -ascii file_name

    where file_name is the output file name you choose. When saving the file, the tool appends the .scf ending for a constraint file in ASCII format.

---

**Importing an ASCII Constraint File**

You can import an ASCII file of constraint commands for the current chip by using menu commands or the scripting tool.

To use GUI commands to import an ASCII constraint file for application to the current implementation,

1. Right-click the elaborated implementation in the Chips window.

2. Choose Edit Constraints from the pop-up menu.

   Steps 1 and 2 set the current chip and enable the menu items in step 3.


   The Import Constraints File dialog box opens.

4. Enter the file name and click OK.
To use the shell to import an ASCII file of constraint commands for the current implementation,

1. Run the scripting tool.
   
   See “Using the Shell” on page 3-4 for instructions.

2. Set the current implementation. At the shell prompt, enter

   current_chip my_chip

3. At the shell prompt, enter

   import_constraint -ascii -f file_name

   where file_name is the name of the file of constraint commands to apply to the current implementation.

When importing a constraint file, the tool skips any lines that contain errors and issues an error message. For debugging the constraint file, a line number appears with each error message. Lines with errors are not processed.

Messages are issued at the command line for the following problems:

- The constraint is unknown (misspelled, for example).
- The constraint is not supported for the object. For example, the architecture does not support the constraint, or the constraint name is misspelled.
- The target object cannot be found. For example, the object name is misspelled.
- The constraint is outside the valid range.
• The object is invalid for the constraint. For example, the object and constraint both exist, but are inappropriate for each other.
FPGA Scripting Tool (FST)

FPGA Scripting Tool (FST) implements a Tcl-based command-line interface to all the synthesis and optimization features of FPGA Express.

This appendix covers the following topics:

- Introduction
- Project Variables
- FST Commands
Introduction

FST is designed for users who

- Prefer command-line or keyboard-based interfaces to graphical interfaces.
- Understand the capabilities of the tool and want to specify operations as quickly as possible. For example, a specification that requires multiple mouse clicks and keyboard entries might be entered as a single command.

The data models defined by the GUI are preserved in command line form by FST. Command-line conventions established by the PrimeTime and Design Compiler products are followed as long as they are consistent with this data model.

Note:
To protect the integrity of encrypted IP cores, several FST commands are disabled for implementations with these cores. See the Intellectual Property topic in the online Help.

Since FST is used as the command-line interface for FPGA Express, a few features are added to ease migration for users who are moving to FPGA Express. FST is capable of writing simple Design Compiler shell scripts that can be used as a starting point for synthesizing an ASIC version of an FPGA Express chip.

FPGA Express includes a translator to translate Design Compiler scripts to FPGA Express scripts. Due to the differences in technology and methodology between ASICs and FPGAs, it is not a one-to-one translation. However, it serves as a good starting point for fc2_shell users.
This is a command-line only feature. The command to use is `translate_dc_script`.

---

**Project Variables**

Some variables exist only when a project is opened. They are called Project Variables and have names beginning with `proj_`. This section presents a list of all the Project Variables and their options, which are enclosed in braces `{}`.

- `proj_clock_default_frequency = integer`
- `proj_enable_vpp = {yes no (default)}`
- `proj_export_directory = /any/legal/directory`
- `proj_export_timing_constraint = {yes no (default)}`
- `proj_fsm_coding_style = {onehot (default) binary zeroonehot}`
- `proj_fsm_optimize_case_default = {yes (default) no}`
- `proj_gsr_ignore_unlinked_cells = {yes no (default)}`
- `proj_xnfin_bus_style = {%s<%d> (default) any string}`

---

**FST Commands**

This section contains lists of commands for working with constraints, generating reports, browsing design objects, managing designs, and so on.

---

**Constraints Commands**

Constraints commands let you import constraints from a file, export constraints to a file, and set constraints.

- `export_constraint`
- `import_constraint`
- `set_cell_max_fanout`
set_chip_advanced_opt
set_chip_constraint_driven
set_chip_effort
set_chip_gsr_ignore_unlinked_cells
set_chip_hierarchy
set_chip_lpm
set_chip_max_fanout
set_chip_mem_map
set_chip_objective
set_chip_primitive
set_chip_sharing
set_module_dont_touch
set_module_effort
set_module_hierarchy
set_module_mem_map
set_module_objective
set_module_primitive
set_module_rmdup_cells
set_module_sharing
set_open_drain
set_pad_buffer
set_pad_dir
set_pad_input_voltage
set_pad_loc
set_pad_output_voltage
set_pad_reg_delay
set_pad_register
set_pad_resistance
set_pad_slew_rate

---

**Reports Commands**

Reports commands let you generate reports about chips, files, libraries, and projects.

report_chip
report_file
report_library
report_project
Browsing Objects Commands

Object browsing commands let you traverse objects in a design.

get_cell
get_chip
get_clock
get_design
get_device
get_file
get_library
get_module
get_net
get_pathgroup
get_pin
get_port
get_register
get_speed
get_target

Timing Commands

Timing commands let you create and delete subpaths, get timing information, set the clock speed, and set delays.

create_subpath
delete_subpath
report_timing
set_disable_timing
set_clock
set_input_delay
set_max_delay
set_output_delay

Source Design Management Commands

The list_design command lets you list all design names.
list_design

---

**Target Management Commands**

The list_target command lets you list all design targets.

list_target

---

**Logical Library Management Commands**

Logical library management commands let you create libraries, delete libraries, and generate lists of libraries.

create_library
delete_library
list_library

---

**Chip Management Commands**

Chip management commands let you read SDF files, create chips, list the current chip, delete chips, export chips, generate lists of chips, optimize chips, set the chip export directory, and so on.

chip_read_sdf
create_chip
current_chip
delete_chip
export_chip
list_chip
optimize_chip
script_chip
set_chip_export_directory
set_chip_retiming
Source File Management Commands

Source file management commands let you add design source files, analyze files, generate lists of files, and delete files.

add_file
analyze_file
list_file
remove_file

Project Management Commands

Project management commands let you open a project, close the current project, and create a project.

close_project
create_project
open_project

Built-In Tcl Commands

The FPGA scripting language includes many commands in the Tcl standard.

after
alias
append
apropos
array
break
catch
cd
clock
close
concat
continue
define_proc_attributes
echo
eof
error
error_info
eval
exec
exit
expr
fblocked
fconfigure
file
fileevent
flush
for
foreach
format
gets
glob
global
help
history
if
incr
info
interp
join
lappend
lindex
linsert
list
llength
load
lrange
lreplace
lsearch
lsort
man
open
package
parse_proc_arguments
pid
print_suppressed_messages
printvar
proc
puts
pwd
quit
read
redirect
regexp
regsub
return
scan
seek
set
socket
source
split
string
subst
suppress_message
switch
tell
time
trace
unalias
unset
unsuppress_message
update
uplevel
upvar
vwait
which
while

Other Commands

This section list other miscellaneous FST commands.

list_message
list_status
translate_dc_script
update_chip
update_file
update_library
update_project
active edge
The active edge of a clock signal at a sequential element is the edge at which the output signal becomes stable.

For flip-flops, the active edge is the clocking edge. For example, the rising edge is the active edge for rising-edge-triggered flip-flops.

For level-sensitive latches, the active edge is the end of the enabled time. For example, the falling edge is the active edge for positively enabled latches.

Active edge is an important concept for understanding how default timing values are assigned to path groups.

analyze
What the tool does to check for syntax errors and verify that the HDL source file contents conform to Synopsys HDL policy.

In the GUI, the HDL design source files are automatically analyzed when you add them to a library. The shell commands for adding and analyzing design source files are add_file and analyze_file.

See also design source files.

analysis order
The order in which design source files are analyzed; that is, the files are analyzed in the order in which you add them to a project.
Analysis order is important only for VHDL files and for files in which packages are defined and then referenced. For example, if a package defined in file1 uses another package defined in file2, add file2 before adding file1. This order of addition causes the tool to analyze file2 before file1. Be sure to add design source files to the project in the order in which they must be analyzed.

See also design source files and project.

clock
A signal with a periodic behavior. In synchronous circuit designs, clocks are used to synchronize the propagation of data signals by controlling sequential elements. It is important that you accurately specify all clocks so that a synchronous circuit can be optimized efficiently. The tool automatically analyzes the circuit and lists all the signals that require clock definitions in the Clocks constraint table.

A clock is defined by its waveform, with rising and falling edge times in the interval between 0 and its period. Because the interval always starts at time 0, the precise relationship between all the clocks in the design are known by the tool.

A clock can also be specified by its frequency, in megahertz. In this case, the tool converts the frequency into a waveform rising at time 0 and falling at the specified fall time.

delay
The time it takes a signal to propagate from one point to another. All delays specified are in nanoseconds and must be expressed as integers.

design source files
Text files that contain the design description, which can be VHDL, Verilog HDL, EDIF, and XNF. Source files can be created using any text editor.
elaboration
The process of mapping a text-based design to technology-independent library cells. Elaboration is part of the Create Implementation process.

During elaboration, logic is inferred from the design source code logic (for example, an if-then-else statement is translated into an AND-OR network, a “+” is translated into an addition operator). Only designs in files that are analyzed can be elaborated.

See also analyze.

ending group of a timing path
The ending group of a path can be the set of all primary outputs of the design, all edge-sensitive sequential elements clocked by a specified periodic signal, or all level-sensitive sequential elements clocked by a specified periodic signal.

See also starting group.

flip-flop
An edge-triggered device controlled by periodic signals (clocks). Flip-flops synchronize the propagation of data signals.

See also clock.

input delay
The input delay of an input or inout port is the maximum delay from that port to a timing group.

global buffer
Global buffers drive clocks or high fanout nets to make a design faster, minimize clock skew, or make the routing task easier. Some buffers can drive all nets, others can only drive clock nets. Buffers are architecture-dependent. Global buffers are usually a limited resource.
Global buffers are available only to designs that use Actel or Xilinx technologies and that do not have the Do Not Insert I/O Pads option selected.

See also *global buffer insertion*.

**global buffer insertion**
The process of inserting global buffers into the netlist to minimize clock skew and make designs faster and easier to route. Global buffer insertion takes into account architecture-dependent issues such as the number of buffers available for a particular architecture and the ability of a global buffer to drive nonclock signals. Global buffer insertion is part of the optimization process.

See also *global buffer*.

**global set/reset (GSR)**
A signal that asynchronously sets or resets all of the sequential elements in a design.

See also *sequential elements*.

**global set/reset (GSR) inferencing**
A built-in optimization that automatically detects the presence of global set/reset (GSR) signals in your design. The signal is marked with a target technology-specific GSR marker block. Place and route tools use this marker to identify the GSR signal and route it using dedicated routing.

Automatic GSR signal detection and dedicated routing assignment help improve the performance of your design.

**hand-instantiated pad**
A primitive pad cell that is instantiated by the designer in the netlist or HDL description is defined as a hand-instantiated pad cell.

**HDL library**
A collection of designs that are stored in the tool.
**inout port**
A port whose net is driven by a three-state driver and loads internal logic or other ports in the design.

See also *three-state driver*.

**input register delay**
The setup time for registers mapped to input pads or driven directly by input ports.

The shell command for specifying the use of an input delay inside an input register during optimization is `set_pad_reg_delay`.

**latch**
An enabled device usually controlled by a periodic signal. Latches allow propagation of data signals during a specific time interval. Latches are also called *level-sensitive devices*.

**module**
A module is part of an implementation hierarchy and is the container for its submodules. A module’s label is the name of its top-level (root) design. The top-level design of an implementation is associated with the top level design of a hierarchy.

See also *module boundary* and *top-level design*.

**module boundary**
A module has a boundary defined by its ports. Module boundaries can be preserved or eliminated during optimization. You can use the Preserve and Eliminate settings in the Hierarchy column of the Modules constraint table to control whether boundaries are preserved during optimization.

**multicycle path**
A design might have path groups that are longer than a single cycle. To make a path group multicycle, change the default path delay computed by FPGA to the new delay representing the multicycle behavior. For example, if a path group is clocked at both ends by 20
ns clocks and its default path delay computed from the clock waveforms is 10 ns, changing the default delay from 10 ns to 30 ns will add one more cycle to this path group.

**netlist files**
FPGA netlist files are text files that contain the design netlist description. These files are in EDIF or XNF format.

**output delay**
The output delay of an output or inout port is the maximum delay from a timing group to that port.

**pad constraints**
When you specify pad constraints, you describe the type of pad cells that the pad-mapping optimization inserts at the ports of the top-level design. You can specify these pad characteristics: port pad type, resistance, input register delay, I/O registers, input voltage, and slew rate.

See also *input register delay*.

**pad mapping**
The optimization step that creates the I/O pads at each port in the top-level design. This optimization is controlled by the pad constraints and by the global pad controls you set when you create implementations.

See also *top-level design* and *pad constraints*.

**path delay**
Path delay for a path group is the maximum delay allowed from any point in the starting timing group to any point in the ending timing group.

**Note:**
To protect the integrity of encrypted IP cores, the Paths constraint table is disabled for implementations that include these cores.
**path group**
A path group is the set of combinational paths from one timing group (called the *starting group*) to another timing group (called the *ending group*).

**point-to-point timing**
A timing point is defined as either the input or output of a register or a primary I/O of the design.

**primitive**
A basic technology library unit used in optimized netlists. For information about the primitives in a specific architecture, see the documentation for the technology library.

You can also instantiate primitives in the HDL description of a design by using named association.

**project**
Is a directory (folder) created by the tool. This working directory holds a project file with an .exp extension and subdirectories created by the tool for internal and intermediate use. You do not need to look at the contents of the subdirectories; they are automatically maintained.

See also *project file*.

**project file**
The project file with an .exp extension. This file contains all the information that the tool needs to reopen an existing project.

See also *project*.

**project state**
The project state is defined by the status of the design source files and the status of the implementation. (See Source file status icons and the Implementation status icons in the online Help).
**resistor pad**
A pull-up or pull-down cell that causes the logic value of a three-state net to be either logic 1 or logic 0 when the net’s three-state drivers are all driving a logic z or high impedance.

**sequential elements**
Flip-flops and latches are collectively referred to as sequential elements.

See also *flip-flop* and *latch*.

**slack**
Slack is the margin by which a delay requirement is met. Positive slack means that the requirement is met; negative slack means that the requirement is not met.

**starting group of a timing path**
The starting group of a path can be the set of all primary inputs of the design, all edge-sensitive sequential elements clocked by a specified periodic signal, or all level-sensitive sequential elements clocked by a specified periodic signal.

See also *ending group of a timing path*.

**three-state output port**
A port whose net is driven by a three-state driver and does not load other logic or ports in the design.

See also *three-state driver*.

**timing group**
A timing group is a collection of sequential elements and ports in the top-level design that have common timing behavior.

These are timing groups:
- All input and inout ports belong to the INPUT group
- All output and inout ports belong to the OUTPUT group
- All flip-flops clocked by the same edge of a common clock belong to a group
- All latches enabled by the same value of a common clock belong to a group
- All flip-flops or latches enabled by the same value of a common signal belong to a group

A timing group is a central concept, because all timing-constraint specifications use timing groups as basic entities.

top-level design
An HDL design can be partitioned into a set of smaller subdesigns. This partitioning produces a hierarchy of designs. The top-level design refers to the design at the top of this hierarchy.

See also design source files.
Index

A
active edge, defined GL-1
adding design source files 5-3
analysis order, defined GL-1
analyze, defined GL-1

B
back-annotation 1-7
boundary, module, defined GL-5
buffer, global, defined GL-3
buffer, insertion of global buffers, defined GL-4
bus styles
  XNF 4-8

C
chip, current, setting 12-11
clock frequency, setting 6-4
clock, defined GL-2
clocks constraint table 10-2
commands
  current_chip 3-6, 12-2
  export_constraint 3-6
  export_netlist 3-6
  import_constraint 12-12
  translate_dc_script A-3
constraint file
  ASCII, exporting 12-10
  ASCII, importing 12-12
  error checking 12-12
  subpath caution note 12-7
  subpaths 12-6
  working with 12-1
constraint tables 10-2
  editing cells in 7-4
  how to open 7-3
constraint-driven design flow 2-4
constraints
  comparing to actual results 8-2
  entering 7-1
  pad, defined GL-6
context-sensitive Help 3
creating a new project 5-2
creating source files 4-1
current chip 3-6
current chip, setting 12-11
current project 3-6
current_chip command 3-6
D

databases, project 11-5
debugging source files 5-6
delay
  input register delay, defined GL-5
  output, defined GL-6
  path, defined GL-6
delay extraction 10-2
delay, defined GL-2
delay, input, defined GL-3
design constraints
  comparing to actual results 8-2
  entering 7-1
design flow 1-2, 2-1
  constraint-driven 2-4
debugging source files 5-6
DesignWizard 2-2
  HDL-based 1-3
  hierarchical (multiple-device) 2-5
  push-button 2-3
  script-based 2-4
  synthesizing a design 6-2
design implementation 6-3
design methodology (see design flow)
design source files
  adding libraries of 5-3
  adding to project 5-4
  defined GL-2
  updating from project 5-8
design, top-level, defined GL-9
DesignWizard, using 2-2

element, sequential, defined GL-8
ending group of a timing path, defined GL-3
error checking, constraint file, when importing 12-12
errors and warnings 5-6
export_constraint command 3-6
export_netlist command 3-6
exporting
  constraints 12-1

F

files
  design source files, defined GL-2
  netlist files, defined GL-6
  project, defined GL-7
flip-flop, defined GL-3
FPGA Express, defined 1-2
FPGA scripting tool (FST) 3-5
FPGA scripting tool (FST) (see scripts)
FST, FPGA scripting tool 3-5

generating
  project databases 11-5
generating netlists 11-2
generating reports 11-3
global buffer insertion, defined GL-4
global buffer, defined GL-3
global set/reset (GSR) inferencing, defined GL-4
global set/reset (GSR), defined GL-4
group, path, defined GL-7
group, timing, defined GL-8
GSR (global set/reset) inferencing, defined GL-4
GSR (global set/reset), defined GL-4
H
hand-instantiated pad, defined GL-4
HDL design methodology, migrating to 1-3
HDL editor 5-6, 5-7
HDL library, defined GL-4
HDL modules 4-8
help
  context-sensitive online Help 3
  Output window 3-2
  Synopsys Technical Support Center xxii
  tip bar 3-2
  topic-based online Help 3
hierarchical (multiple-device) design flow 2-5

I
identifying top-level design 6-2
implementation
  current chip, setting the 12-11
design, creating 6-3
import_constraint command 12-12
importing
  constraints 12-1
inferencing, global set/reset (GSR), defined GL-4
inout port, defined GL-5
input delay, defined GL-3
input register delay, defined GL-5
insertion of global buffers, defined GL-4

K
known problems xvii

L
latch, defined GL-5
libraries of design source files 5-3
library, HDL, defined GL-4

M
man pages 7
mapping, pad, defined GL-6
methodology, using DesignWizard 2-2
module boundary, defined GL-5
module, defined GL-5
modules constraint table 10-4
mouse behavior 3-2
multicycle path, defined GL-5
multiple-device design flow 2-5

N
Netlist Module 4-8
netlists
  files, defined GL-6
  generating 11-2

O
optimization
  how to accomplish 8-1
  viewing results 10-2
output delay, defined GL-6
output port, three-state, defined GL-8
Output window
  description 3-2
  figure of 5-6
  information in 5-6
pad constraints, defined GL-6
pad mapping, defined GL-6
pad, hand-instantiated, defined GL-4
pad, resistor, defined GL-8
path
delay, defined GL-6
multicycle, defined GL-5
path group, defined GL-7
path, timing
ending group, defined GL-3
starting group, defined GL-8
Paths constraint table 10-2, 10-5
place and route 1-7, 7-2
point-to-point timing, defined GL-7
port, inout, defined GL-5
port, three-state output port, defined GL-8
Ports constraint table 10-3
primitive, defined GL-7
proj_export_directory variable 8-8
proj_fsm_coding_style variable 8-6
proj_fsm_optimize_case_default variable 8-7
project
defined GL-7
file, defined GL-7
state, defined GL-7
variables A-3
project variables A-3
project window 5-5
project, setting up 5-1
push-button design flow 2-3

register delay, input, defined GL-5
Release Note, viewing xvii
reports, generating 11-3
resistor pad, defined GL-8

sample scripts 3-4
schematic viewer 1-2, 9-1
script-based design flow 2-4
scripts 1-2, 2-4
sequential element defined GL-8
setting clock frequency 6-4
simulation 1-4, 1-7, 11-2, 11-3
slack, defined GL-8
source files
debugging 5-6
design source files defined GL-2
starting group of a timing path, defined GL-8
state, project, defined GL-7
subpath
cautery note 12-7
specifying in constraint file 12-6
Support Center, Synopsys Technical xxii
synthesizing a design 6-2

table cells, editing 7-4
three-state output port, defined GL-8
TimeTracker 10-5
timing analysis 10-2
timing group, defined GL-8
timing path
ending group, defined GL-3
starting group, defined GL-8
timing, point-to-point, defined GL-7
tip bar 3-2
topic-based online Help 3
top-level design
defined GL-9
identifying 6-2
top-level design, identifying 6-2
translate_dc_script command A-3
U
updating source files 5-8

V
variables A-3
  proj_export_directory 8-8
  proj_fsm_coding_style 8-6
  proj_fsm_optimize_case_default 8-7
viewing errors and warnings 5-6

viewing results of optimization 10-2
viewing schematics
  mapped 9-5
  optimized 9-5
  RTL or generic 9-4
viewing timing results 10-5

X
XNF 5-1, 5-3
XNF bus style 4-8