Foundation Series 2.1i Quick Start Guide

Foundation Series 2.1i Quick Start Guide — 0401832 *Printed in U.S.A.*

About This Manual

This guide should be used as the initial learning tool for designers who are unfamiliar with the features of the Foundation series software.

Additional Resources

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

Resource	Description/URL
Tutorial	Tutorials covering Xilinx design flows, from design entry to verification and debugging http://support.xilinx.com/support/techsup/tutorials/index.htm
Answers Database	Current listing of solution records for the Xilinx software tools Search this database using the search function at http://support.xilinx.com/support/searchtd.htm
Application Notes	Descriptions of device-specific design techniques and approaches http://support.xilinx.com/apps/appsweb.htm
Data Book	Pages from *The Programmable Logic Data Book*, which describe device-specific information on Xilinx device characteristics, including read-back, boundary scan, configuration, length count, and debugging http://support.xilinx.com/partinfo/databook.htm
Xcell Journals	Quarterly journals for Xilinx programmable logic users http://support.xilinx.com/xcell/xcell.htm
Tech Tips	Latest news, design tips, and patch information on the Xilinx design environment http://support.xilinx.com/support/techsup/journals/index.htm

Manual Contents

This guide covers the following topics.

- **Chapter 1**, "Setting Up the Foundation Tools," gives instructions for installing Foundation 2.1i and provides you with information about the type of computer you need to successfully implement your designs.
- **Chapter 2**, "Foundation Overview," looks in-depth at the capability and flexibility of the Foundation software.
- **Chapter 3**, "Basic Tutorial" provides a step-by-step example explaining how to use the basic Foundation tools.
- **Appendix A**, "Glossary," defines some of the commonly used terms in this Guide.

Conventions

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

Typographical

The following conventions are used for all documents.

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

  ```
  speed grade: -100
  ```

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

  `rpt_del_net=`

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

  `File` → **`Open`**

- *Italic font* denotes the following items.

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

    `edif2ngd` *design_name*

  - References to other manuals

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

- Emphasis in text

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

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

  edif2ngd [*option_name*] *design_name*

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

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

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

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

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

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

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

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

Online Documents

The following conventions are used for online documents.

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

Contents

Chapter 3 Basic Tutorial

Appendix A Glossary

Chapter 1

Setting Up the Foundation Tools

This chapter lists the system requirements for the Foundation Series 2.1i Xilinx design tools software and discusses the recommended machine types and memory requirements to comfortably run the software. Also included are general instructions for installing the software, contacting customer support, and obtaining and installing the necessary authorization codes and licenses.

For a detailed discussion, refer to the "System Requirements" chapter in the *Foundation Series 2.1i Installation Guide and Release Notes.*

This chapter contains the following sections:

- "Installation Notes"
- "Customer Service"
- "Technical Support"

Installation Notes

Ensure the optimum use and operation of your new design tools by installing Foundation Series 2.1i on the recommended hardware with sufficient memory (RAM and hard disk "swap" space). If you experience problems with either the installation, operation, or verification of your installation, contact the Xilinx Technical Support hotline. Refer to the "Technical Support" section of this chapter for specifics.

Supported Platforms and Machine Requirements

The Foundation Series 2.1i software is a PC-only release. Foundation runs on either Windows 95/98 or Windows NT. (Service Pack 3 or 4 required with NT.) The following list shows the minimum recommended type of PC you should have to perform designs for Xilinx FPGAs or CPLDs.

- Pentium Pro Processor®
- Windows 95®, Windows 98®or Windows NT 4.0® (with Service Pack 3 or 4 installed)
- 120 MHz clock speed
- System Memory—32 MB to 64 MB (dependent on device)
- Swap Space—48 MB to128 MB (dependent on device)
- Required disk space, 2 GB recommended
- SVGA 17" monitor
- 4x CD-ROM drive
- Ports—Two ports (one for a pointing device and one parallel port for the parallel download cable, if needed). You can share the parallel port used for the parallel download cable.
- Keyboard
- Mouse—2-button or 3-button (Microsoft Windows compatible). On a 3-button mouse, the middle button is not used.

Note: Due to the size and complexity of the XC4000 and Virtex devices, Xilinx recommends that these designs be compiled using a high-performance computer. 64 MB of RAM as well as 64 MB of swap space is required to compile XC4000EX designs, but Xilinx recommends that at least 128MB of both RAM and swap space be used. For Virtex designs, Xilinx recommends 256 MB of RAM.

Swap file size requirements also vary with the design and constraint set size. By default, Windows 95/98 manages its swap file size automatically, but for Windows NT, you may need to increase it. Typically, your Windows NT swap file size should be twice as large as your system RAM amount.

It is important to note that slower systems or systems with less than the recommended RAM and/or swap space may exhibit longer runtimes.

Memory Requirements for Xilinx Architectures

The various steps of designing Xilinx FPGAs or CPLDs require a substantial amount of memory, as shown in the following table.

Table 1-1 Minimum Memory Requirements

Xilinx Packages	RAM	Virtual Memory (Swap Space)
Base or Base Express	48 MB	64 MB
Standard or Foundation Express	64 MB	128 MB

Note: The values given in the above table are for typical designs and include the normal load created by the operating system. Additional memory may be required for certain “boundary-case” or “extremely large” designs, as well as for concurrent operation of other non-Xilinx applications.

Running Setup

Please refer to the *Foundation Series 2.1i Installation Guide and Release Notes* for complete details on installation and prerequisites for installation.

1. To start the installation, insert the Design Environment CD into the CD-ROM drive.

   If your system has the Auto Run feature enabled, the Foundation setup program will start automatically.

   If you do not have the Auto Run enabled on your system, select **`Start`** → **`Run`**. Type **`d:setup.exe`** in the Open field of the Run window and click **`OK`**. (If your CD-ROM drive is not the "d" drive, substitute the appropriate drive designation.)

   You are required to enter your name, company, and CD key in the User Information screen before you can begin the installation. Your CD Key is the number printed on the CD Key label on the back of the CD case. The Product Information label, which is also located on the back of the CD case, contains the part number and serial number.

2. Follow the instructions on the screen to install the software. When complete, remove the CD.

3. (Optional) To install the Xilinx documentation CD, insert the CD and follow the instructions.

You may need to reboot your PC to allow the new / modified environment variables and path statement to take effect before you can run the design implementation tools. The Install program will inform you if you need to reboot.

There are optional CDs that you may have received in your package. Consult the *Foundation Series 2.1i Installation Guide and Release Notes* for information about these CDs.

Network Compatibility

The Xilinx installation program supports only TCP-IP style networks. Novell is *not* a TCP-IP style network. You can run the Xilinx implementation tools from a network but not the Aldec design entry tools or Express. Also, ABEL cannot be run from the network.

Obtaining and Setting Up Licenses

A license is only required for the Base Express and Foundation Express products. New and existing customers installing the Foundation Series 2.1i Base or Standard products do *not* need Express licenses.

For complete details about licensing, refer to the "Express Software Licensing" chapter in the *Foundation Series 2.1i Installation Guide and Release Notes*.

If you are a new customer with no license.dat file installed and no LM_LICENSE_FILE variable set, a temporary license is automatically installed with Base Express and Foundation Express. With the Foundation Express temporary license, the Express constraints editor, timing analyzer, and Schematic Viewer GUIs are not available.

When you install the Foundation 2.1i software, this temporary license file (license.dat) is copied to %XILINX%\data on your system. In addition, the LM_LICENSE_FILE file variable is automatically set up to point to this license.dat file. When you receive a permanent license.dat file, Xilinx recommends that you put this permanent license outside the Xilinx software tree. A common location is c:\flexlm.

To obtain a permanent license, you need to be a registered user in the Customer Service database.

New Xilinx users should fill out their Xilinx registration card and *fax or mail* it to their Customer Service location. Customer Service will send your license and authorization codes. You can also register online within the software install program.

If you are an existing customer and you need a new license, you can obtain the new license by accessing online Web registration at http://support.xilinx.com. If you request your license by fax, please fill out the form that is enclosed in your package.

Upon receiving a permanent license from Xilinx, make sure your LM_LICENSE_FILE environment variable points to the new license file.

The following scenarios describe the requirements for licensing. For details about these scenarios and other licensing issues, see the

"Express Software Licensing" chapter in the *Foundation Series 2.1i Installation Guide and Release Notes*.

- If you are an existing customer upgrading from Base or Standard to Base Express or Foundation Express, you need to obtain a new license.dat file from Xilinx.
- If you are an existing customer with Base Express and are upgrading to 2.1i Base Express, you must modify the package definition in your license.dat file. Please see the "Upgrading an Existing License" section.
- If you are an existing customer with Foundation Express and are upgrading to 2.1i Foundation Express, you must modify the package definition in your license.dat file. Please see the "Upgrading an Existing License" section.
- If you are an existing customer with Base Express and are upgrading to Foundation Express, you must obtain a new license.dat file from Xilinx.
- If you are an existing customer with Base or Standard and are upgrading to the 2.1i Base or Standard, you do not need to obtain a license, that is, there are no licensing or registration requirements.
- If you are a new customer receiving Base or Standard, register with Xilinx. You do not need to obtain a license.
- If you are a new customer receiving Base Express or Foundation Express, you must register with Xilinx and obtain a permanent license.dat file.

Upgrading an Existing License

If you are upgrading Base Express or Foundation Express to 2.1i, you must manually update your existing license file. To update this file, perform the following steps:

1. Locate the existing permanent Foundation license file. This file is typically located at C:\flexlm\license.dat and will have FND BSX-PC or FND-EXP-PC increment lines similar to the following:

```
INCREMENT FND-BSX-PC xilinxd 1.000 01-JAN-0 0 CCD4DEC6B7D7723BB082\
"XSJ_davet" 00a024a9ea43


INCREMENT FND-EXP-PC xilinxd 1.000 01-JAN-0 0 DC24BEE6B3D04455B07E  \
"XSJ_davet" 00a024a9ea43
```

2. Open the license.dat file using a plain text editor.
3. Replace the existing PACKAGE definitions with the following PACKAGE definitions.

   The package definitions are available in electronic format in a file named "license_update.txt". The license_update.txt file is located both on the Foundation Series 2.1i CD and is copied to the Foundation Series 2.1i installation area in %XILINX%\data.

   - Foundation Series 2.1i CD

     Insert the CD in the CDROM drive (assume D:\). The package definitions are located in D:\license_update.txt.

   - Foundation Series 2.1i installation area

     The package definitions are located in C:\FNDTN\data\license_update.txt

For FND-BSX-PC

```
PACKAGE FND-BSX-PC xilinxd 1.000 D07010C1B82E60A3A701 \
        COMPONENTS="system-PC bit-PC \
        xc3000D-PC xc4000E-PC xc5200E-PC \
        ngd2vhdl-PC verilog-PC \
        Foundation-PC X-VHDL-PC \
        FPGA-Express:2000.05 \
        FPGA-Express-VHDL-Base:2000.05 \
        FPGA-Express-VLOG-Base:2000.05 \
        FPGA-Express-XC3k-Optimizer:2000.05 \
        FPGA-Express-XC4k-Optimizer:2000.05 \
        FPGA-Express-XC5k-Optimizer:2000.05 \
        FPGA-Express-XC9k-Optimizer:2000.05 "
```

For FND-EXP-PC

```
PACKAGE FND-EXP-PC xilinxd 1.000 2080B0F13916AA26C238 \
        PACKAGE FND-EXP-PC xilinxd 1.000 20405041295F375FE6F6 \
        COMPONENTS="system-PC bit-PC \
        xc3000D-PC xc4000X-PC xc5200X-PC \
        ngd2vhdl-PC verilog-PC \
        Foundation-PC X-VHDL-PC \
        FPGA-Express:2000.05 \
        FPGA-Express-VHDL-Base:2000.05 \
        FPGA-Express-VLOG-Base:2000.05 \
        FPGA-Express-XC3k-Optimizer:2000.05 \
        FPGA-Express-XC4k-Optimizer:2000.05 \
        FPGA-Express-XC5k-Optimizer:2000.05 \
        FPGA-Express-VIRTEX-Optimizer:2000.05 \
        FPGA-Express-XC9k-Optimizer:2000.05 \
        FPGA-Express-Constraint-Mgr:2000.05 \
        FPGA-Express-GAT:2000.05 "
```

Customer Service

For software licensing information, warranty status, shipping, and order management issues, contact Xilinx Customer Service using the information in the following table.

Country	Telephone	Facsimile
United States and Canada[1]	1-800-624-4782	408-559-0115
United Kingdom[2]	01932-333550	01932-828521
Belgium[2]	0800 73738	
France[2]	0800 918333	
Germany[2]	0130 816027	
Italy[2]	1677 90403	
Netherlands[2]	0800 0221079	
Other European Locations[2]	(44) 1932-333550	(44) 1932-828521
Japan	81 3 3297 9153	81 3 3297 9189

[1]Mon-Fri, 8:00 am - 5:00 pm Pacific time

[2]Monday–Friday, 9:00 a.m. to 5:30 p.m. United Kingdom time—English speaking only.

If you are an international customer, contact your local sales representative for customer service issues. Refer to the Xilinx web site at http://support.xilinx.com/company/sales/int_reps.htm for contact information.

A complete list of Xilinx worldwide sales offices is at http://support.xilinx.com/company/sales/offices.htm.

Technical Support

The following section details how to reach the Xilinx Application Service centers for your area. If you experience problems with the installation or operation of your software, Xilinx suggests that you first go to our http://support.xilinx.com website.

You can also contact the Xilinx Technical Support hotline by phone, email, or fax. When e-mailing or faxing inquiries, provide your complete name, company name, and phone number. The following table gives Worldwide contact information for Xilinx Application Service centers.

Location	Telephone	Electronic Mail	Facsimile (Fax)
North America	1-408-879-5199 1-800-255-7778	hotline@xilinx.com	1-408-879-4442
United Kingdom	44-1932-820821	ukhelp@xilinx.com	44-1932-828522
France	33-1-3463-0100	frhelp@xilinx.com	33-1-3463-0959
Germany	49-89-93088-130	dlhelp@xilinx.com	49-89-93088-188
Japan	local distributor	jhotline@xilinx.com	local distributor
Korea	local distributor	korea@xilinx.com	local distributor
Hong Kong	local distributor	hongkong@xilinx.com	local distributor
Taiwan	local distributor	taiwan@xilinx.com	local distributor
Corporate Switchboard	1-408-559-7778		

Chapter 2

Foundation Overview

This overview explains the basic concepts and design flow of the Foundation Series 2.1i release as it spans the flow from netlist to final PROM file. The chapter describes the basic tools; for details on using the tools, refer to the "Basic Tutorial" chapter.

The Foundation Overview chapter contains the following sections:

- "New Features"
- "Design Flows"
- "Using the Foundation Design Entry Tools"
- "Using the Design Implementation Tools"
- "Using Constraint Files"
- "Static Timing Analysis"
- "Creating Simulation Files"
- "Downloading a Design"
- "Re-Entrant Routing (FPGAs)"

The flow described in this chapter is generally applicable to all Xilinx families. However, many of the details apply only to the FPGA device families. For complete information on CPLD design flows, refer to the Foundation online help.

New Features

The major new features for the Foundation 2.1i release include the following:

- Integration of the CORE Generator tool within the Project Manager
- Enhanced project data archiving
- New 3.2 version of Foundation Express
- Guide file functionality
- New conversion utilities, ABEL2HDL and AHDL2HDL
- Language Assistant support for the Core Generator tool
- Schematic support in HDL flows
- New device support
- Integration of the Xilinx Constraints Editor in the Project Manager
- Runtime improvement
- CPLD ChipViewer
- New HTML document viewer for all online documents

For a detailed description of these new features, refer to the What's New file by selecting **`Start`** → **`Programs`** → **`Xilinx Foundation Series 2.1i`** → **`What's New`**.

For detailed information about Xilinx documentation, refer to the "Introduction" chapter of the *Foundation Series 2.1i User Guide.*

Design Flows

The Foundation Series design tools interface supports two basic flows within the Project Manager: HDL and Schematic.

HDL Flow

An HDL Flow project can contain VHDL, Verilog, or schematic top-level designs with underlying VHDL, Verilog, or schematic modules.

HDL files can be created using the HDL Editor, Finite State Machine Editor, or other text editors. Design sources are analyzed and optimized by the Express Synthesis Engine.

LogiBLOX, CORE Generator, and ABEL modules as well as XNF files can be instantiated in the design source files using the "black box instantiation method". Black box modules are not elaborated and optimized during synthesis. State machine modules are synthesized as VHDL or Verilog.

For a detailed description of the design methodologies, refer to the "Design Methodologies - HDL Flow" chapter in the *Foundation Series 2.1i User Guide*.

Schematic Flow

The Schematic Flow supports the following design strategies.

- Top-level schematic design with the Xilinx Unified Libraries components, LogiBLOX symbols, CORE generated modules, and ABEL, HDL and/or state machine macros
- Top-level ABEL-based designs (not recommended for FPGA designs)
- Top-level State Machine designs—only as ABEL designs.

For a detailed description of the design methodologies, refer to the "Design Methodologies - Schematic Flow" chapter in the *Foundation Series 2.1i User Guide*.

Also refer to the "HDL Design Entry and Synthesis" chapter in the *Foundation Series 2.1i User Guide*.

The following two figures illustrate the basic design flow for FPGAs and CPLDs. For detailed design flow illustrations, refer to the "File Processing Overview" appendix in the *Foundation 2.1i Series User Guide*.

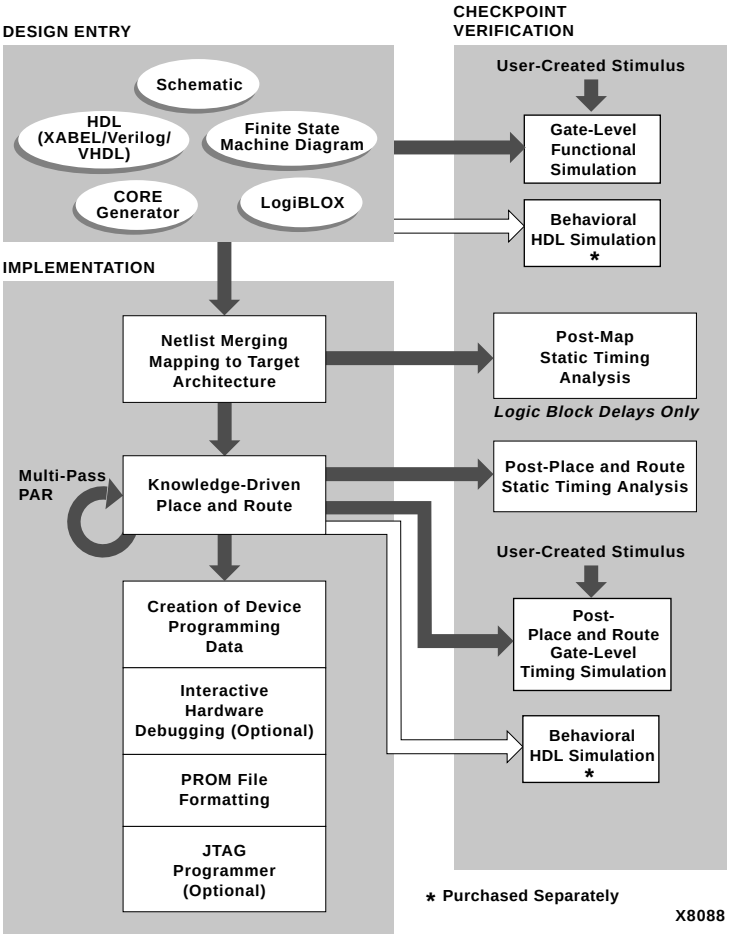


Figure 2-1 Foundation Overall Design Flow for FPGAs

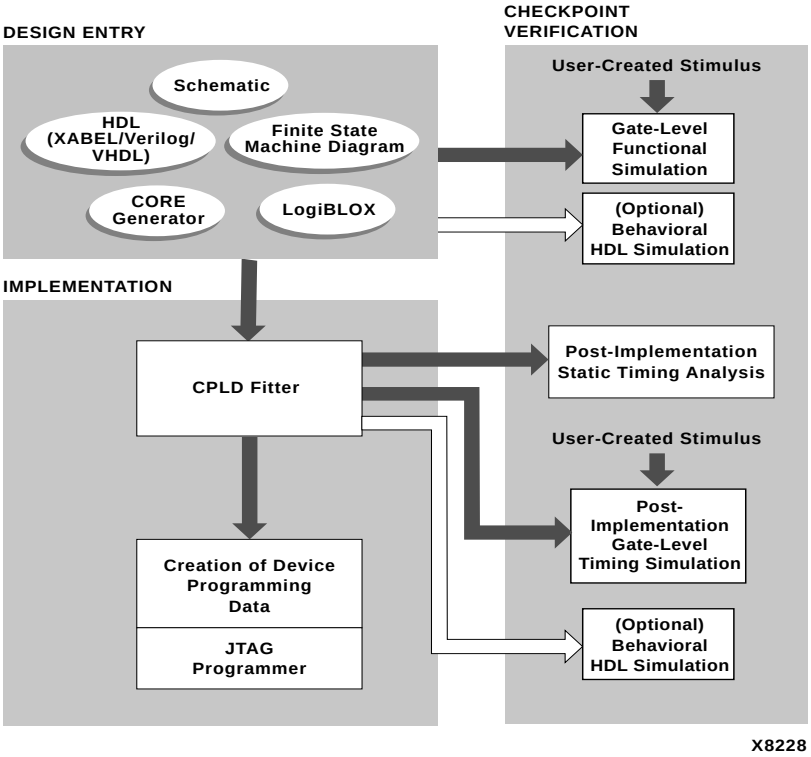


Figure 2-2 Foundation Overall Design Flow for CPLDs

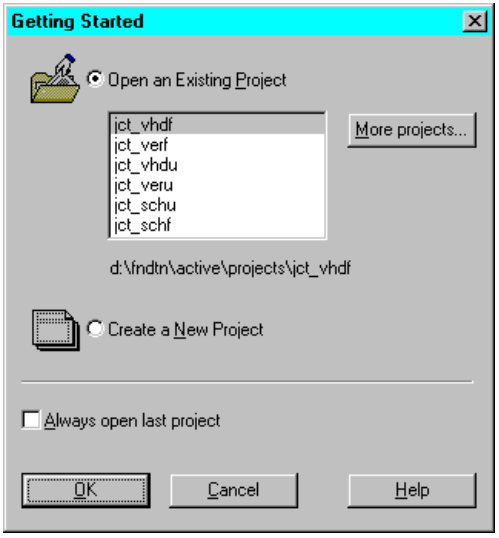
Using the Foundation Design Entry Tools

This section describes the basic procedure for using the design entry tools.

Starting the Foundation Project Manager

To start the Project Manager, double click the Project Manager icon in the Foundation Series program group. The icon to click is shown in the following figure.

A Getting Started dialog box displays, allowing you to select a project to open or create a new project.



Creating a New Project

To create a new project, follow these steps:

1. Select Create a New Project and click **OK**.

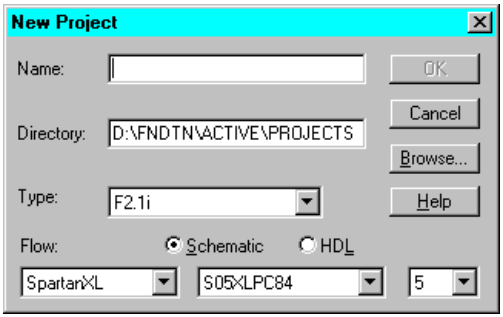


Figure 2-3 New Project Dialog Box

2. After the New Project dialog box displays, enter a name for the project. Change the directory for the project, if desired, by using the Browse button.
3. Choose the appropriate family, part, and speed grade and flow type. (schematic flow only)
4. Click **OK**. The new project displays in the Project Manager.

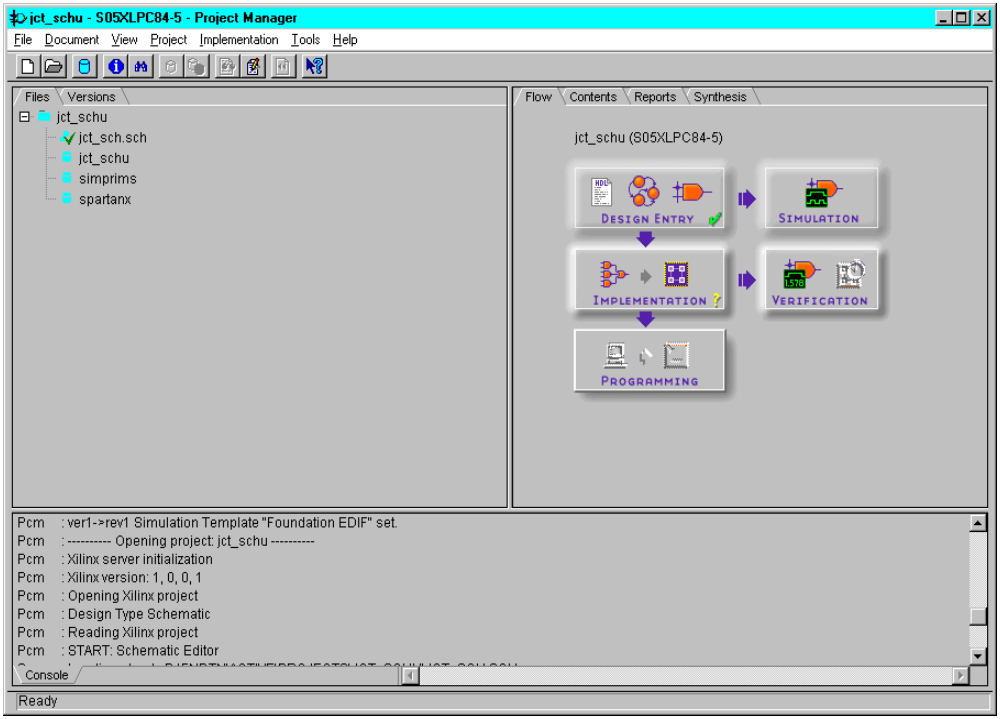


Figure 2-4 Foundation Project Manager

For more information about creating new projects in Foundation, refer to Foundation's online help system. For detailed information about the Project Manager, refer to the online help by selecting **Help** → **Foundation Help Contents** → **Project Manager**. Also see the "Project Manager" section of the "Project Toolset" chapter in the *Foundation Series 2.1i User Guide*.

Creating Top-level VHDL/Verilog Designs

You can create VHDL and Verilog designs if you purchased a version of Foundation that includes the Synopsys FPGA Express package. FPGA Express is included with these Foundation versions: 1) Base Express, and 2) Foundation Express.

You can create a variety of top-level schematic, VHDL or Verilog designs.

- All-HDL designs
- HDL designs with State Machine macros
- HDL designs with black box instantiations
- Schematic designs

Black boxes are not synthesized by Express; they are passed to the implementation tools for translation by the Flow Engine.

For a detailed description of the procedures for creating these types of designs, refer to the "Design Methodologies - HDL Flow" chapter in the *Foundation Series 2.1i User Guide*.

Also refer to the Foundation 2.1i Watch Tutorial located at http://support.xilinx.com/support/techsup/tutorials/index.htm.

For a discussion of HDL design issues, refer to the "HDL Design Entry and Synthesis" chapter in the *Foundation Series 2.1i User Guide*.

For information on how to use the VHDL and Verilog languages, refer to the online software documents, *VHDL Reference Guide* and *Verilog Reference Guide*.

Foundation 2.1i also includes utilities that convert ABEL and AHDL to HDL (ABEL2HDL and AHDL2HDL). To access these utilities from the Project Manager, select **Tools** → **Utilities**. In order to use ABEL or AHDL designs as top-level HDL designs, you must convert the designs.

Creating Top-Level Schematic Designs

You can create a variety of top-level designs.

- All schematic designs
- Schematic designs with instantiated HDL macros, LogiBLOX and CORE Generator modules, and state machine macros
- Top-level ABEL-based designs
- State Machine designs—only as ABEL designs

For a detailed description of the procedures for creating these types of designs, refer to the "In-Depth Tutorial—Schematic-Based Designs" in the *Foundation 2.1i Watch Tutorial* located at http://support.xilinx.com/support/techsup/tutorials/index.htm.

Also refer to the "Design Methodologies - Schematic Flow" chapter in the *Foundation Series 2.1i User Guide.*

For a discussion of schematic design issues, refer to the "Schematic Design Entry" chapter in the *Foundation Series 2.1i User Guide.*

Creating State Machine Designs

State machine designs typically start with the translation of a concept into a "paper design," usually in the form of a state diagram or a bubble diagram. The paper design is converted to a state table and then, into the source code itself.

A State Machine design can be used in the following ways.

- Top-level design in a Schematic Flow—as an ABEL design only
- A module in a schematic
- A module in a VHDL or Verilog design (not ABEL)

For a detailed discussion of the design steps, refer to the "Design Methodologies - Schematic Flow" chapter and the "Design Methodologies - HDL Flow" chapter in the *Foundation Series 2.1i User Guide*.

For a description of a sample state machine, refer to the "State Machine Designs" chapter in the *Foundation Series 2.1i User Guide.*

Instantiating LogiBLOX and CORE Generator Modules

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

For information about instantiating LogiBLOX into designs, refer to the following sections in the *Foundation Series 2.1i User Guide*:

- "Schematic Designs With Instantiated LogiBLOX Modules" section of the "Design Methodologies - Schematic Flow" chapter
- "HDL Designs with Black Box Instantiation" section of the "Design Methodologies - HDL Flow" chapter.

The Xilinx CORE Generator tool is an easy-to-use design tool that delivers parameterizable cores, optimized for Xilinx FPGAs. The CORE Generator library includes cores as complex as DSP filters and multipliers and as simple as delay elements. You can use these cores as building blocks in order to complete your design more quickly. In Foundation 2.1i, the CORE Generator tools are integrated into the Project Manager, Schematic Editor, and HDL Editor. For details on how to instantiate cores in schematics, refer to the "Schematic Designs With Instantiated CORE Generator Cores" section of the "Design Methodologies - Schematic Flow" chapter. For details on how to instantiate cores in HDL designs, refer to the "CORE Generator COREs in a VHDL or Verilog Design" section of the "Design Methodologies - HDL Flow" chapter of the *Foundation Series 2.1i User Guide*.

For complete information about the CORE Generator tool, refer to the online manual, *CORE Generator System User Guide*.

Accessing the Design Entry Tools

You can access all of the Design Entry tools from the Project Manager's Tools menu (**`Tools`** → **`Design Entry`**). The tools include the following:

- Schematic Editor
- State Editor
- HDL Editor
- Symbol Editor
- LogiBLOX module generator
- CORE Generator

You can also directly access the Schematic Editor, State Editor, and HDL Editor from the Design Entry phase button.

For a complete description on how to use these design entry tools, see the *Foundation Series 2.1i User Guide.*

After completing a design with design entry, the design is next implemented to a target Xilinx hardware part.

Using the Design Implementation Tools

The implementation tools perform the translate map, place, route, (fit for CPLDs), and bitstream generation phases of the design flow

The Xilinx Flow Engine is the graphical interface that displays each of of these design phases. Results of these implementations are made available in reports and may be accessed through the Reports tab in the Project Manager.

The Foundation Project Manager provides menu and pushbutton access to other Xilinx tools such as the Timing Analyzer, PROM File Formatter, Floorplanner, FPGA Editor, Constraints Editor, and CPLD ChipViewer. You can access these tools through either the Tools menu or Implementation menu in the Project Manager or directly from the Project Flowchart pushbuttons.

During design implemenation, the Flow Engine prominently displays the status of each phase of the design, as shown in the following figures.

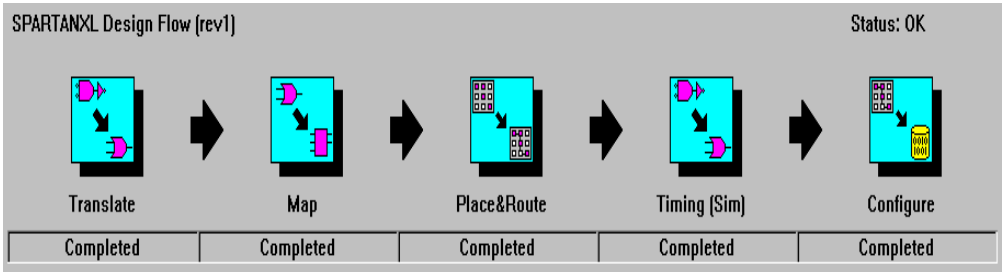


Figure 2-5 Flow Engine Shows All Design Segments Completed (FPGAs)

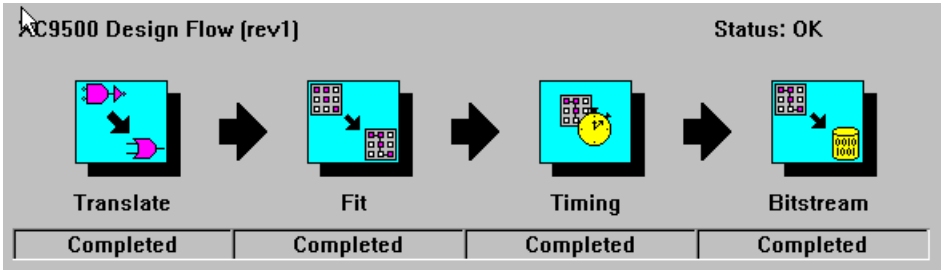


Figure 2-6 Flow Engine Shows All Design Segments Completed (CPLDs)

Translate

The Flow Engine's first step, Translate, merges all of the input netlists. This is accomplished by running NGDBuild. For a complete description of NGDBuild, refer to the "NGDBuild" chapter of the *Development System Reference Guide*.

MAP (FPGAs)

The next step is the technology mapper. Map optimizes the gates and trims unused logic in the merged NGD netlist. This step also maps the design's logic resources; logic in the design is mapped to resources on the silicon, and a physical design rule check is performed. For more information about MAP, refer to the "MAP—The Technology Mapper" chapter in the online software manual, *Development System Reference Guide*.

Place and Route (FPGAs)

After the design is mapped, the Flow Engine places and routes the design. In the place stage, all logic blocks, including the configurable logic blocks (CLB) and input/output blocks (IOB) structures, are assigned to specific locations on the die.

If timing constraints have been placed on particular logic components, the placer tries to meet those constraints by moving the corresponding logic blocks closer together.

In the routing stage, the logic blocks are assigned specific interconnect elements on the die. If timing constraints have been placed on particular logic components, the router tries to meet those constraints by choosing a faster interconnect. For more information about PAR, refer to the "PAR—Place and Route" chapter in the online software document, *Development System Reference Guide*.

CPLD Fitter (CPLDs)

The CPLD fitter implements designs for the XC9500 and XC9500XL devices. The fitter outputs several files: fitting report (*design_name*.rpt), static timing report (*design_name*.tim), guide file (*design_name*.gyd, programming file (*design_name*.jed), and timing simulation database (*design_name*.nga).

For detailed information about implementing CPLD designs, refer to the Foundation online help.

Configure (FPGAs)

After place and route, the Flow Engine translates the physical implementation into a configuration file (bit) that is used to program the FPGA. The BitGen executable creates the configuration file. For more information about the BitGen executable, refer to the "BitGen" chapter in the online software document, *Development System Reference Guide*.

You can program an FPGA using the Hardware Debugger or JTAG Programmer to download a bitstream to configure a device. You can also use a bitstream as an input to the PROM File Formatter, which creates a specific configuration program for PROM use.

Bitstream (CPLDs)

At the end of a successful CPLD implementation, a .jed programming file is created. The JTAG Programmer uses this file to configure XC9500/XL CPLD devices.

Interpreting the Reports

The reports generated by the implementation tools provide information on logic trimming, logic optimization, timing constraint performance, and I/O pin assignment. To access the reports, select the Reports tab from Project Manager. Double click the Implementation Report Files icon to open the Report Browser. To open a particular report, double click its icon, as shown in the "Report Browser (FPGAs)" figure.

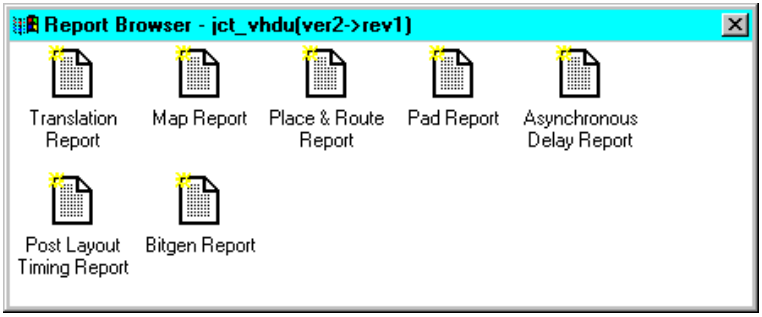


Figure 2-7 Report Browser (FPGAs)

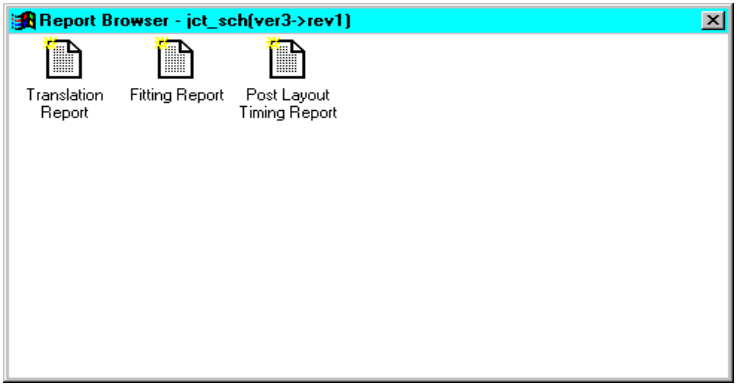


Figure 2-8 Report Browser (CPLDs)

Translation Report

The translation report (.bld) contains warning and error messages from the three translation processes: conversion of the EDIF or XNF style netlist to the Xilinx NGD netlist format, timing specification checks, and logical design rule checks. The report lists the following:

- Missing or untranslatable hierarchical blocks
- Invalid or incomplete timing constraints
- Output contention, loadless outputs, and sourceless inputs

Map Report (FPGAs)

The Map Report (.mrp) contains warning and error messages detailing logic optimization and problems in mapping logic to physical resources. The report lists the following information:

- Removed logic. Sourceless and loadless signals can cause a whole chain of logic to be removed. Each deleted element is listed with progressive indentation, so the origins of removed logic sections are easily identifiable; their deletion statements are not indented.
- Logic that has been added or expanded to optimize speed.
- The Design Summary section lists the number and percentage of used CLBs, IOBs, flip-flops, and latches. It also lists occurrences of architecturally-specific resources like global buffers and boundary scan logic.

Note: The Map Report can be very large. To find information, use key word searches. To quickly locate major sections, search for the string '---' , because each section heading is underlined with dashes.

Place and Route Report (FPGAs)

The Place and Route Report (.par) contains the following information.

- The overall placer score which measures the "goodness" of the placement. Lower is better. The score is strongly dependent on the nature of the design and the physical part that is being targeted, so meaningful score comparisons can only be made between iterations of the same design targeted for the same part.

- The Number of Signals Not Completely Routed should be zero for a completely implemented design. If non-zero, you may be able to improve results by using re-entrant routing or the multi-pass place and route flow.
- The timing summary at the end of the report details the design's delays. For information on timing constraint performance and synchronous delays, refer to the "Static Timing Analysis" section later in this chapter.

Pad Report (FPGAs)

The Pad Report lists the design's pinout in three ways.

- Signals are referenced according to pad numbers.
- Pad numbers are referenced according to signal names.
- PCF file constraints are listed.

Fitting Report (CPLDs)

The Fitting Report *(design_name*.rpt) lists summary and detailed information about the logic and I/O pin resources used by the design, including the pinout, error and warning messages, and Boolean equations representing the implemented logic.

Post Layout Timing Report

A timing summary report shows the calculated worst-case timing for the logic paths in your design.

Asynchronous Delay Report

This report shows the 20 worst net delays within the design.

Selecting Options

Options specify how a design is optimized, mapped, placed, routed, and configured. Options are grouped into objects called implementation, simulation, and configuration templates. Each template defines an implementation, simulation or configuration approach. For example, one implementation style could be Quick Evaluation, while another could be Timing Constraint Driven.

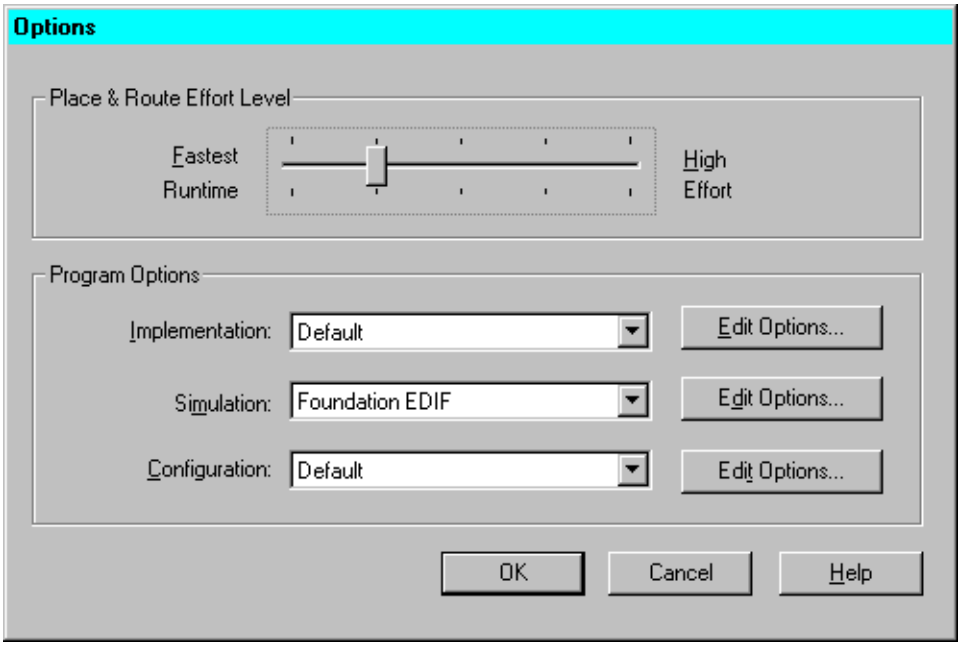


Figure 2-9 Flow Engine Options Dialog Box

You can have multiple templates in a project. To access the options and templates, perform the following steps.

1. Select the Options button in the Implement or Synthesis / Implementation dialog box.
2. In the Program Option portion of the Options dialog box, select the Edit Template button for Implementation, Simulation, or Configuration to access the associated template.

The default options settings provide sufficient performance for most design requirements. For information on the options, select **Help** → **Help Topics** from the Flow Engine menu.

Using Constraint Files

With the design implementation tools, you can control the implementation of a design by entering constraints. There are two basic types of constraints that you can apply to a design: location constraints and timing constraints.

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

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

Design, Netlist, and User Constraints

Constraints can be entered throughout the design entry and implementation processes. Constraints can be entered during the design entry phase by adding them to a schematic, specifying them through the use of a constraint entry GUI, or listing them in a user constraint file (.ucf). These three approaches differ in the following ways.

- Constraints entered directly in the input design are known simply as design constraints and are ultimately placed in the design netlist.
- If you want your constraints separated from the input design files, or if you want to modify your constraints without having to completely re-synthesize your design, you can use the Xilinx Constraints Editor or create a user constraints file *design_name*.ucf.

Using the Xilinx Constraints Editor

The Constraints Editor is a Graphical User Interface (GUI) that you can run after the Translate program to create new constraints in a UCF file. To access the Constraints Editor, select **Tools** → **Implementation** → **Constraints Editor** from the Project Manager.

The Constraints Editor interface consists of a main window, three tab windows, and a number of dialog boxes. For more details, refer to the online software document, *Constraints Editor Guide*.

You can also directly enter constraints into a UCF file without using the Xilinx Constraints Editor. See the next section for details.

Creating a User Constraint File

The user constraint file (.ucf) is a user-created ASCII file that holds timing and location constraints. It is read by NGDBuild during the translate process and is combined with an EDIF or XNF netlist into an NGD file. If a UCF file exists with the same name as the top-level netlist, then it will automatically be read. Otherwise, specify a file for User Constraints in the Implement Control Files Settings dialog box.

For Foundation 2.1i, if you already have an existing UCF file associated with a Revision, this UCF file is automatically copied and used as your UCF file within a new revision.

For an example of how to lock I/Os to pin locations and how to write Timespec and Timegroup constraints, refer to the “Foundation Constraints” appendix.

You can also lock pin locations within the Project Manager by selecting **Tools** → **Implementation** → **Lock Device Pins**.

Static Timing Analysis

You can perform timing analysis at several stages in the implementation flow to estimate delays. You create or generate the following.

- A post-synthesis pre-implementation display for HDL designs with the Express Time Tracker (Foundation Express only—not Base Express).
- A post-map timing report to evaluate the effects of logic delays on timing constraints.
- A post-place-and-route timing report that incorporates both block and routing delays as a final analysis of the design's timing constraints.

The Interactive Timing Analyzer tool produces detailed timing constraint, clock, and path analysis for post-map or post-place-and-route implementations.

Note: Static timing analysis may make the implementation processes run slower.

Static Timing Analysis after Synthesis (HDL Only)

You can examine static timing results with the Express Time Tracker after synthesis and before implementation. You must be licensed to use Foundation Express to access the Time Tracker and the Express Constraints Editor.

1. After you synthesize your design, right click the optimized structure from the Versions tab.
2. Select View Synthesis Results.
3. Select the Paths tab from the Time Tracker to view estimated delays.

Static Timing Analysis after Map (FPGAs Only)

Post-map timing reports can be very useful in evaluating timing performance. Although route delays are not accounted for, the logic delays can provide valuable information about the design.

If logic delays account for a significant portion (> 50%) of the total allowable delay of a path, the path may not be able to meet your timing requirements when routing delays are added.

Routing delays typically account for 40% to 60% of the total path delays. By identifying problem paths, you can mitigate potential problems before investing time in place and route. You can redesign the logic paths to use less levels of logic, tag the paths for specialized routing resources, move to a faster device, or allocate more time for the path.

If logic-only-delays account for much less (<35%) than the total allowable delay for a path or timing constraint, then the place-and-route software can use very low placement effort levels. In these cases, reducing effort levels allow you to decrease runtimes while still meeting performance requirements.

Static Timing Analysis after Place and Route (FPGAs Only)

Post-PAR timing reports incorporate all delays to provide a comprehensive timing summary. If a placed and routed design has met all of your timing constraints, then you can proceed by creating configuration data and downloading a device. On the other hand, if you identify problems in the timing reports, you can try fixing the problems by increasing the placer effort level, using re-entrant routing, or using multi-pass place and route. You can also redesign the logic paths to use fewer levels of logic, tag the paths for specialized routing resources, move to a faster device, or allocate more time for the paths.

For information on re-entrant routing or multi-pass place and route, see the “Re-Entrant Routing (FPGAs)” section at the end of this chapter.

Summary Timing Reports

Summary reports show timing constraint performance and clock performance. Implementing a design in the Flow Engine can automatically generate summary timing reports. To create summary timing reports, perform the following steps:

1. Open the Options dialog box (**Implementation** → **Options**) from the Project Manager and select **Edit Options** for the Implementation template.
2. Select the Timing Reports tab.
3. For a post-map report, select **Produce Logic Level Timing Report**. For a post-PAR report, select **Produce Post Layout Timing Report**.
4. To modify the reports to highlight path delays or paths that have failed timing constraints, select a report format.
5. After MAP or PAR has completed, the respective timing reports appear in the Report Browser.

Detailed Timing Analysis

To perform detailed timing analysis, select **Tools** → **Simulation/ Verification** → **Interactive Timing Analyzer** from the Project Manager menu. You can specify specific paths for analysis, discover paths not affected by timing constraints, and analyze the timing performance of the implementation based on another speed grade. For path analysis, perform the following:

1. Choose sources. From the Timing Analyzer menu, select **Path Filters** → **Custom Filters** → **Select Sources**.
2. Choose destinations. From the Timing Analyzer menu, select **Path Filters** → **Custom Filters** → **Select Destinations**.
3. To create a report, select one of the options under the Analyze menu.

To switch speed grades, select **Options** → **Speed Grade**. After a new speed grade is selected, all new Timing Analyzer reports will be based on the design running with new speed grade delays. The design does not have to be re-implemented, because the new delays are read from a separate data file.

Creating Simulation Files

After the design is implemented, you can perform a timing simulation to ascertain if the timing requirements and functionality of your design have been met. Timing simulation can save considerable time by reducing the time spent debugging test boards in the lab. Functional simulation can also potentially save time by uncovering design bugs before running PAR.

When Can Simulation Data be Created?

With the design implementation tools, you can create simulation data after each major processing step. This means that you can create functional simulation netlists after NGDBuild merges the design together in the Translate process and simulation netlists after PAR has placed and routed the design for FPGAs or the CPLD fitter has fit the design for CPLDs.

Additionally, for FPGAs, you can create simulation data after the design has been mapped. For a graphical representation of when you can conveniently simulate your design, refer to the “Foundation Overall Design Flow for FPGAs” figure and the “Foundation Overall Design Flow for CPLDs” figure.

For FPGAs, simulation data created after the design has been mapped contains timing data based on the CLB and IOB block delays, and all net (interconnect) delays are set to zero.

With post-map simulation, you can ensure that the design’s current implementation will give the place and route software sufficient margin to route the design and still stay within your timing requirements.

Simulation data created after the design has been placed, but not routed, contains accurate block delays and estimates for the net delays.

You can use post-place simulation as an incremental simulation step between post-map simulation and a complete post-route timing simulation.

To simulate at any of these intermediate stages, select **Tools** → **Simulation/Verification** → **Checkpoint Gate Simulation Control** from the Foundation Project Manager and choose the appropriate netlist to simulate.

Creating Functional Simulation Data

For schematic and HDL designs, the functional simulation netlists are created in the Foundation design entry tools environment. Click the Simulation phase button in the Project Manager Flowchart area to invoke the Simulator and load the netlist. The Simulation phase button is shown in the following figure.

For designs that include macros whose underlying files are XNF or EDIF netlists, the design must first be "translated" in the Xilinx implementation tools in order to merge in these additional netlists. Follow these steps to translate the design and then invoke the simulator and load the functional netlist.

1. Select **Project** → **Create Version** from the Project Manager.
2. Select **Project** → **Create Revision** from the Project Manager.
3. Select **Tools** → **Implementation** → **Flow Engine** from the Project Manager while the new revision is selected in the Versions tab.
4. From within the Flow Engine, select **Setup** → **Stop After** and then choose the **Stop After Translate** option.
5. Click **OK**, then select **File** → **Run** in the Flow Engine.
6. After Translate is complete, go back to the Foundation Project Manager and select **Tools** → **Simulation/Verification** → **Checkpoint Gate Simulation Control**.
7. Choose the appropriate NGD file from the Revision which was just created and click **OK**. This invokes the simulator and loads the netlist.

For details about functional simulation, refer to the "Functional Simulation" chapter in the *Foundation Series 2.1i User Guide* and the "In-Depth Tutorial—Functional Simulation" chapter in the *Foundation 2.1i Watch Tutorial* located at http://support.xilinx.com/support/techsup/tutorials/index.htm.

For additional information about functional simulation, see the "Performing Functional Simulation" section of the "Design Methodologies - Schematic Flow" chapter in the *Foundation Series 2.1i User Guide*.

Creating Timing Simulation Data

Before you perform timing simulation, ensure that you have generated a timing annotated simulation netlist. See the "Timing Simulation" section of the "Verification and Programming" chapter in the *Foundation Series 2.1i User Guide* for details and the "In-Depth Tutorial—Timing Simulation" chapter in the *Foundation 2.1i Watch Tutorial* located at http://support.xilinx.com/support/techsup/tutorials/index.htm.

1. To create the timing simulation netlist, open the Options dialog box by selecting **Implementation** → **Options**, and verify that Foundation EDIF displays in the Simulation list box for Program Options.
2. Click **OK**.
3. After the Implementation process is complete, return to the Foundation Project Manager, and click the timing simulation portion of the Verification phase button. This invokes the Simulator and loads the timing simulation netlist.

For additional information about timing simulation, refer to the "Verifying the Design" section of the "Design Methodologies - Schematic Flow" chapter in the *Foundation Series 2.1i User Guide*.

HDL Simulation

Foundation provides the option of adding HDL simulation capabilities to all Foundation design flows. Xilinx ships an evaluation version of an HDL simulator from MTI.

This product may be licensed for free evaluation for up to 30 days. Sale and support for this product is provided directly by the vendors.

All ModelSim product sales are handled directly by MTI and its authorized sales affiliates (email sales@model.com).

Customer support is also provided directly by MTI (email support@model.com or call the main number at (503) 641-1340)

Adding MTI's ModelSim product to the Foundation Series design environment enables simulation of VHDL, Verilog HDL or mixed-HDL designs (Verilog and VHDL). Source code debugging, functional simulation, and back-annotated timing simulation are all supported through this integrated solution. The availability of a mixed-language simulation environment offers maximum flexibility to HDL design methodologies which draw on design elements from both Verilog and VHDL.

Downloading a Design

You can download an implemented FPGA design directly from your PC using the Hardware Debugger program with the XChecker cable, JTAG download cable, or MultiLINX cable. No cables are shipped with the Foundation product.

The Hardware Debugger can download a BIT file or a PROM file: MCS, EXO, or TEK file formats. A BIT file contains configuration information for an FPGA device. For more information on using the Hardware Debugger, see the *Hardware Debugger Guide*.

You can download an implemented CPLD design from your PC using the JTAG Programmer. The JTAG Programmer software is used to configure FPGAs and CPLDs and supports both the XChecker and the Parallel Cable III. This is a GUI based program. See the *JTAG Programmer Guide* for details. Also, see the *Hardware User Guide* for information about cable compatibility.

To download an implemented design, click the Programming icon in the Project Flow area.

Creating a PROM

An FPGA or daisy chain of FPGAs can be configured from serial or parallel PROMs. The PROM File Formatter can create MCS, EXO, or TEK style files. The files are read by a PROM programmer that turns the image into a PROM.

A HEX file can also be used to configure an FPGA or a daisy chain of FPGAs through a microprocessor. The file is stored as a data structure in the microprocessor boot-up code.

In-Circuit Debugging

Once a design has been downloaded to an FPGA, snapshots of internal signal states can be captured and read using the Hardware Debugger program with the XChecker cable, JTAG cable, or MultiLINX cable. You can display the signal states as waveforms in the Hardware Debugger.

This capability allows you to test and debug your design in a real-time environment as it interfaces with the other components on your board. You can also control the states of your state machines by controlling when clock edges are sent to your system clock input.

For more information on in-circuit debugging and the Hardware Debugger, see the *Hardware Debugger Guide*.

Re-Entrant Routing (FPGAs)

The place and route software, PAR, has features that allow it to process complex designs that have tight timing requirements and/or are difficult to route. If your design is completely placed and routed but not meeting timing specifications, PAR can start from where it left off and continue re-routing the design to produce an implementation that meets your timing specifications.

As PAR is running, it continually updates the NCD file with its current placement and routing information. As long as an NCD file exists that is at least placed, PAR can used it for re-entrant routing. To initiate re-entrant routing, follow these steps.

1. In the Project Manager, select **Tools** → **Implementation** → **Flow Engine**.
2. In the Flow Engine, select the **Setup** → **FPGA Re-entrant Route** menu.
3. In the Setup Re-entrant Route dialog box, select **Allow Re-Entrant Route**, which enables the re-entrant route options:
   - Optional: If meeting timing specifications is a critical goal, then select Use Timespecs During Re-entrant Route. If meeting timing specifications is not critical, do not select this option, because timing-driven routing takes much longer to process than non-timing-driven routing.
   - Optional: Select the number of re-entrant routing passes to perform. If left in "Auto," PAR will continue to perform routing iterations until either 1) it determines that it is no longer making significant progress, or 2) the design constraints have been fully met.
   - Optional: Select the number of clean-up passes to run. Clean-up passes are run after the "main" routing passes are complete. Two types of clean-up routing passes can be invoked—cost-based and delay-based. The effectiveness of each type depends on the design, device, and constraints of the implementation.
4. Click **OK** (in the Setup Re-entrant Routing dialog box) to submit the options. This causes the Place and Route icon in the Flow Engine to show a loop back arrow and the Re-Entrant route label.

   If you are specifying timing or location constraints, you have the option to relax them to give PAR more flexibility. If you modify the UCF file, you must step backwards with the Flow Engine and re-run Translation in order to incorporate the changes.

Chapter 3

Basic Tutorial

This tutorial describes the features in the Foundation Series release 2.1i. The tutorial is provided in three separate types of projects:

- schematic
- Verilog
- VHDL

The chapter contains the following sections:

- “Getting Started”
- “Schematic Design Entry”
- “HDL-Based Design Entry”
- “Functional Simulation”
- “Implementation”
- “Timing Simulation”

Getting Started

This section guides you through a typical FPGA-based design procedure using a design called ” JCOUNT.” The JCOUNT design targets a SpartanXL device—S05XLPC84-5; however, all of the principles and flows taught are applicable to any Xilinx device family, unless otherwise noted.

In the first part of the tutorial, you will use the Foundation design entry tools to complete the design. The design is composed of flip-flops, buffers, and pads.

Design Description

Throughout this tutorial, the design is referred to as JCOUNT.

The design begins as an unfinished design. After you complete the design, you will simulate it to verify the functionality.

"JCOUNT" is a simple 4-bit Johnson counter. The completed schematic is shown in the following picture.

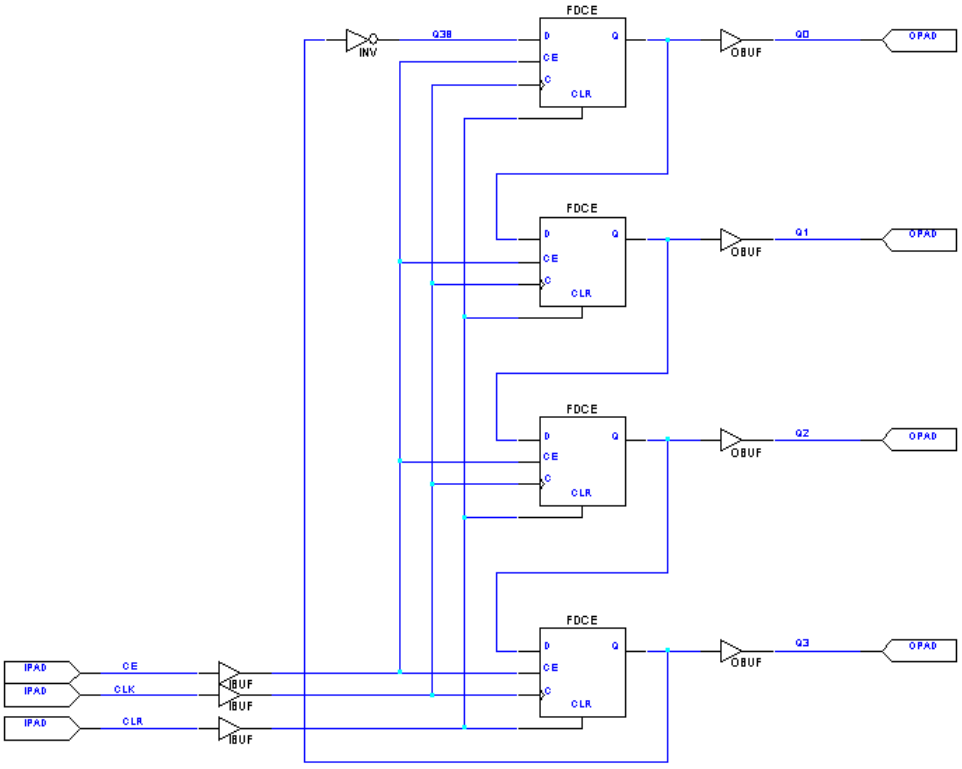


Figure 3-1 Completed JCOUNT Schematic

Controls:

CLK—input clock pulse

CE—clock enable (when set to 0, the counter freezes)

CLR—clear input (when set to 1, the counter is reset to all zeroes)

Outputs:

Q0-Q3—counter outputs

Operation:

The counter is triggered on the rising edge of the clock (CLK) when the clock enable signal (CE) is high. Following is the sequence of states during normal operation (that is, without clearing of the counter):

0000
0001
0011
0111
1111
1110
1100
1000
0000

There are also two files external to the project that contain simulation results.

JCT_F.TVE—prerouted design simulation results

JCT_T.TVE—routed design timing simulation results

Starting the Project Manager

This tutorial assumes that the Foundation software is installed in the default location c:\fndtn. If you have installed the software in a different location, substitute your installation path for c:\fndtn. The JCOUNT project is installed in the c:\fndtn\active\projects directory. Following is the list of JCOUNT projects supplied by Xilinx.

Project Name	Description
JCT_SCHF	JCOUNT schematic -- finished
JCT_SCHU	JCOUNT schematic -- unfinished
JCT_VHDF	JCOUNT VHDL -- finished
JCT_VHDU	JCOUNT VHDL -- unfinished
JCT_VERF	JCOUNT Verilog -- finished
JCT_VERU	JCOUNT Verilog -- unfinished

1. Double click the Foundation Series Project Manager icon on your desktop or select **Start** → **Programs** → **Xilinx Foundation Series 2.1i** → **Xilinx Project Manager** from the Start menu icon in the lower left corner of your screen.

2. A Getting Started dialog box displays, allowing you to select a project to open. If you have not opened this tutorial project before now, click the **More Projects...** button.

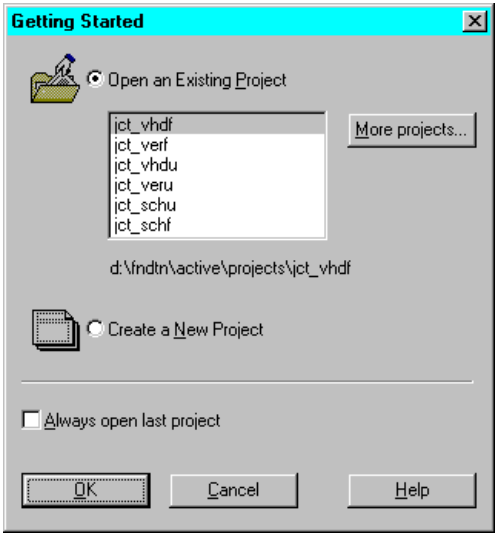


Figure 3-2 Getting Started Dialog Box

3. Browse to the c:\fndtn\active\projects directory in the Directories list (it should open to this location by default). Select one of the following unfinished projects in the Projects list of the Open Project dialog box:
   - JCT_SCHU (JCOUNT schematic design)
   - JCT_VERU (JCOUNT Verilog design)
   - JCT_VHDU (JCOUNT VHDL design)
4. Select **Open** to open the project.

The Project Manager

The Project Manager controls all aspects of the design flow. You can access all of the various design entry and design implementation tools as well as the files and documents associated with your project. The Project Manager also maintains revision control over multiple design iterations.

The Project Manager is divided into three main subwindows. To the left is the Design Hierarchy Browser which displays the project elements. To the right is a set of tabs, each one opens a separate functional window. The third window at the bottom of the Project Manager is the Message Console and shows status messages, errors, and warnings and is updated during all project actions.

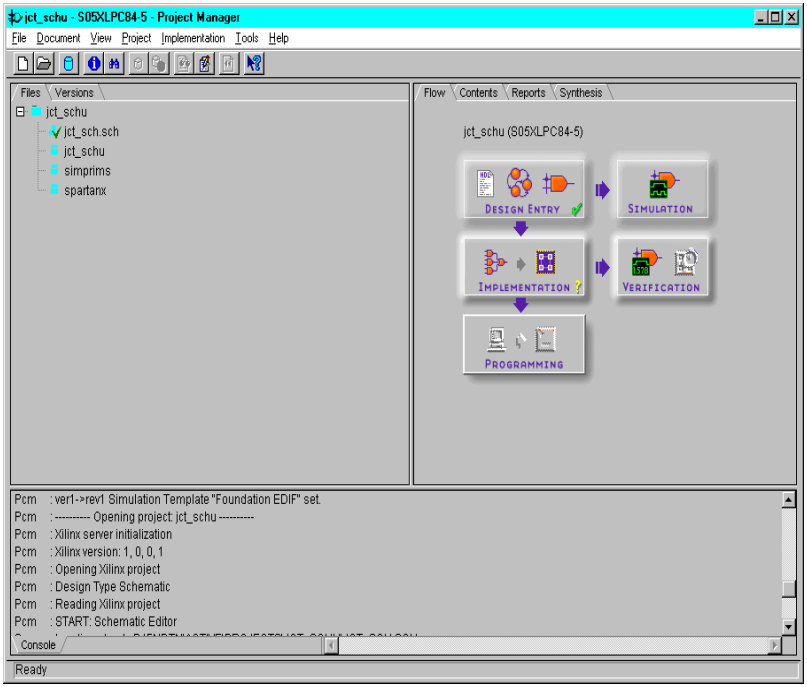


Figure 3-3 Project Manager (Schematic Design)

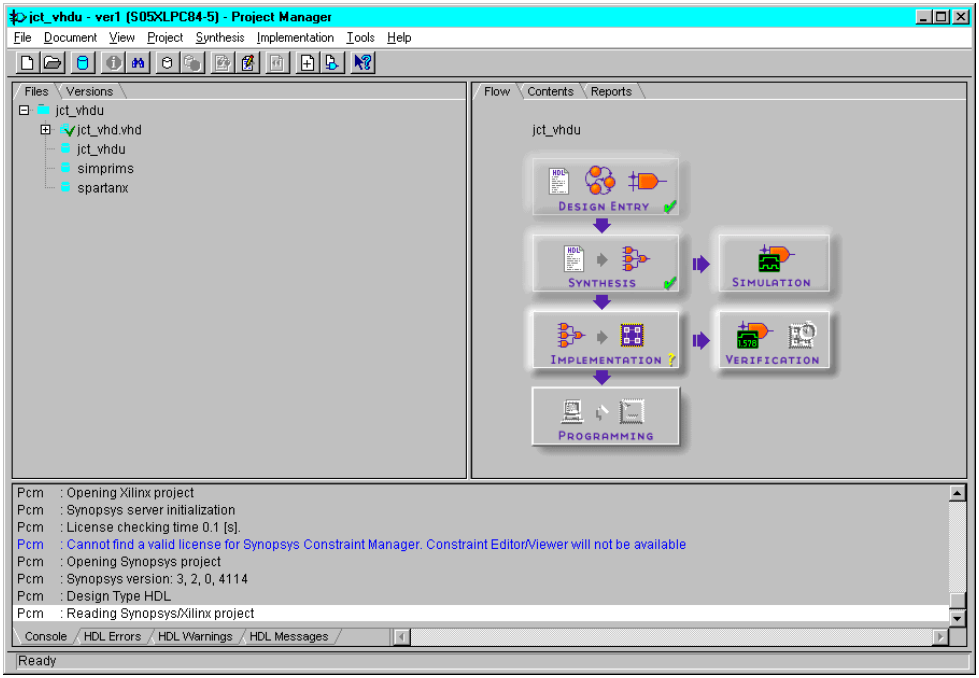


Figure 3-4 Project Manager (HDL Design)

Project Libraries

For schematic designs, when you create a new project in Foundation, three libraries are automatically added to the project: the appropriate device family library based on the target family you have chosen (for example, SPARTANX), the project library (with the same name as the project), and the SIMPRIMS library (for simulation). All libraries that are part of the project are listed in the Files tab of the Project Manager.

For HDL designs, only the project library is initially added to the project. All other libraries are added after synthesis and display in the Files tab.

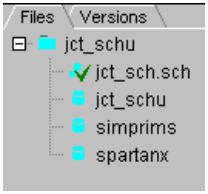


Figure 3-5 Project Libraries (Schematic Design)

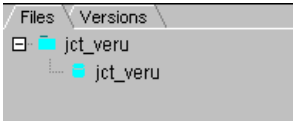


Figure 3-6 Project Libraries (HDL Design—Before Synthesis)

You can double click any of these libraries, which will bring up the Library Manager window, allowing you to see the contents of the library. In the Library Manager window, choose the name of the library (in the Libraries tab), then choose the Objects tab to see which objects are in the particular library.

The device family library (SPARTANX for this project) contains all of the Xilinx Unified Library components for the given family. A complete description of all of these components can be found in the online software manual Xilinx *Libraries Guide.*

To facilitate simulation with the Foundation Logic Simulator, the SIMPRIMS is added to the project. This library contains the simulation models for the Xilinx devices.

To complete an unfinished schematic design, proceed to the next section, “Schematic Design Entry”.

To complete an unfinished HDL design, proceed to the “HDL-Based Design Entry” section.

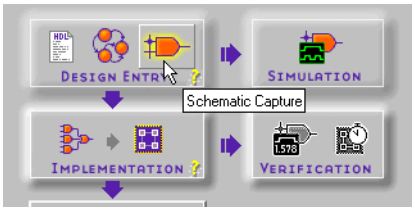
Schematic Design Entry

This section explains how to complete the unfinished schematic design, JCT_SCHU. If you selected an HDL design instead, proceed to the "HDL-Based Design Entry" section.

Starting the Schematic Editor

You can start the Schematic Editor in either of these ways:

- From the Flow tab, click the Schematic Editor icon in the Design Entry phase button. This instructs the Schematic Editor to open the project's top level schematic sheet.



- Double click the file name jct_sch.sch in the Files tab.

The Schematic Editor opens with the jct_sch.sch schematic sheet loaded. The jct_sch.sch schematic is incomplete at this point. You will now complete it.

If you need to stop the tutorial at any time, save your work on the schematic by selecting **`File`** → **`Save`** from the schematic pulldown menus.

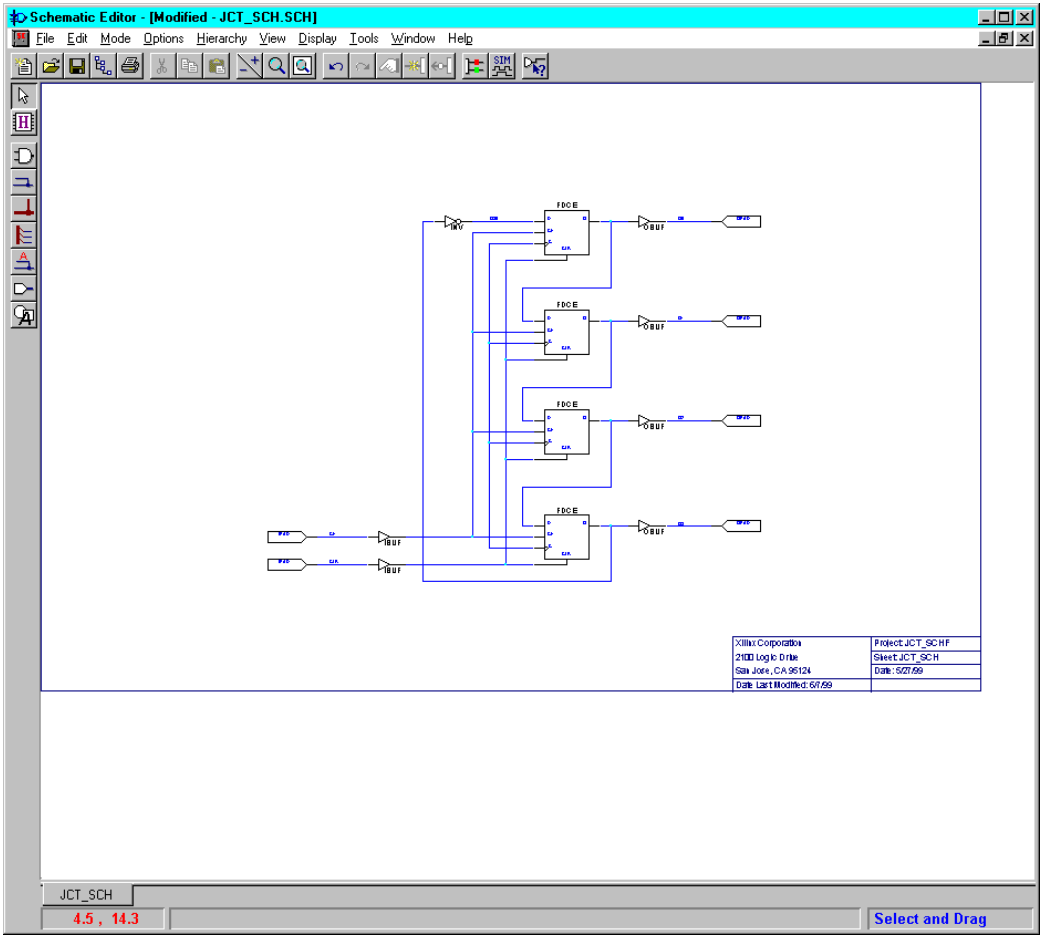


Figure 3-7 Schematic Editor

Manipulating the Screen

Under the Display pulldown menu is a series of commands that modify the viewing area of the Schematic Editor window. Zoom in or out of the schematic to comfortably view it.

Adding a Library Component

Components from all of the libraries (except SIMPRIMS) for the given project are available from the SC (Schematic) Symbols Toolbox to place on the schematic. The available components listed in this toolbox are arranged alphabetically within each library.

1. From the pulldown menus of the schematic editor, select **Mode** → **Symbols** or click the Symbols Toolbox button in the vertical toolbar on the left side of the Schematic Editor.

This puts the schematic editor in "Symbols Mode," opens the SC Symbols window, and displays the libraries and their corresponding components.

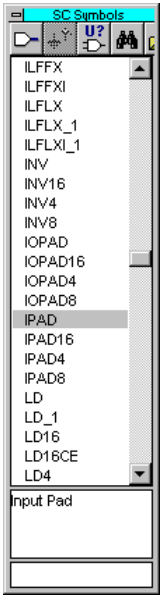


Figure 3-8 SC Symbols Toolbox

2. The component you will place is an IPAD input pad. You can select this component by either scrolling down the list and selecting it or by typing **IPAD** in the bottom of the SC Symbols

Window. Then, move the mouse back into the schematic window.

In the SC Symbols window, when the IPAD component is selected, a description of the component appears in the bottom of the window.

3. Move the symbol outline to the location shown at the lower left of the following picture and click the left mouse button to place the IPAD between the CE and CLR IPADs.

   Remove the SC Symbols window by toggling the Symbols Toolbox icon on the vertical toolbar ("un-click" to exit the toolbox mode).

   You will wire up the IPAD later in the tutorial. For now, just place it on the schematic.

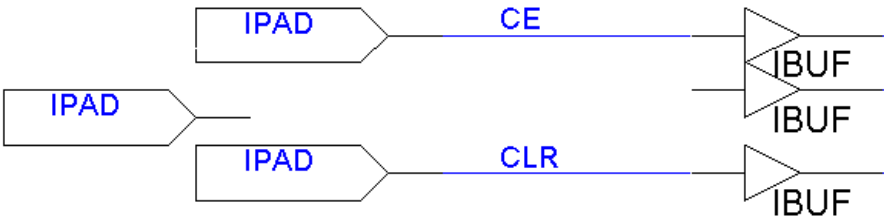


Figure 3-9 Placing IPAD

Correcting Mistakes

If you make a mistake when placing the component, you can easily move or delete the component.

1. Press the **Esc** key on the keyboard to exit the Symbols Mode.
2. Select the component you want to move or delete. Make sure that no other components are selected (clicking a blank area of the schematic deselects everything).

3. Click and drag to correctly place the component, or press the **Del** key on the keyboard or the Cut icon in the toolbar to delete the component.

Drawing and Labeling Nets

You use the Draw Wires icon in the vertical toolbar to draw wires (also called nets) to connect the IPAD to other components on the schematic.

1. Click the Draw Wires icon in the vertical toolbar.

2. Click the source pin (output pin of the IPAD), then click the destination pin (the IBUF to the right of the IPAD that connects to the flip-flop clock). The net will automatically be drawn between the two pins.

Note: You can specify the shape of the net by moving the mouse in the direction you want to draw the net and then single-clicking to create a 90-degree bend in the wire. If you make a mistake while drawing a net, you can quit your current drawing by pressing the "Escape" key.

3. Press the "Escape" key to return to Select-and-Drag mode.
4. Double click the net that you just drew.
5. In the Net Name field, type **CLK** as shown below.

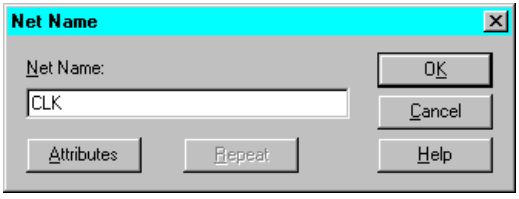


Figure 3-10 Labeling Nets

6. Click **OK**. Now, view your net, labeled, on the schematic sheet.

Saving the Schematic

The jct_sch.sch schematic is now complete.

Save the schematic by selecting **File** → **Save** or clicking the Save icon in the horizontal toolbar.

All errors, warnings, and informational messages are displayed in the Message Window in the Project Manager. If any errors are issued, resolve them and save the schematic again.

Proceed to the "Functional Simulation" section.

HDL-Based Design Entry

In this section, you will use either one of two HDL projects: **jct_veru** or **jct_vhdu**. This is the same JCOUNT design that is used in the schematic design entry. However, rather than being comprised of modules on a schematic sheet, this JCOUNT design is written in VHDL or Verilog.

This section assumes that you have already opened one of the following HDL example designs from the Project Manager.

- `c:\fndtn\active\projects\jct_veru`
- `c:\fndtn\active\projects\jct_vhdu`

Adding a File to the Design

For this tutorial, you will add either the Verilog file (.v) or the VHDL file (.vhd) to the design. Each design has a syntax error. To add the file to the design and fix the errors, perform the following steps:

1. Add the Jct_ver.v or the Jct_vhd.vhd file to the project by selecting **Document** → **Add**. After the Add Document dialog box displays, enter the Jct_ver.v or Jct_vhd.vhd file name in the File name list box and click **Open**.

   Note that the file has a red X next to it. The red X means that the file has an error.

2. Double click the Jct_ver.v or the Jct_vhd.vhd in the Files tab of the Project Manager file to open it.

Correcting Syntax Errors

1. Check the syntax of the HDL file you are using by selecting **`Synthesis`** → **`Check Syntax`** from within the HDL Editor. Both the Jct_ver.v and Jct_vhd.vhd file have the same syntax error—the CLK port is not declared.

   - If you are using the Jct_ver.v file, add the CLK definition to Line 1. Line 1 should look like the following after adding the CLK definition.

     module JCT_VER (CLK, CE, CLR, Q);

   - If you are using the Jct_vhd.vhd file, add the CLK declaration to the Port declaration. The Port declaration lines should look like the following after adding the CLK declaration.

     port (CLK: in STD_LOGIC;
     CE: in STD_LOGIC;
     CLR: in STD_LOGIC;
     Q: inout STD_LOGIC_VECTOR (3 downto 0));

2. Recheck the syntax. The designs now check successfully.

Using the Language Assistant

While you are in the HDL editor, view the Language Assistant. The Language Assistant provides templates (example code fragments) for commonly used HDL constructs, as well as synthesis templates for commonly used logic components such as counters, D flip-flops, multiplexers, and global buffers. You can add your own templates to the Language Assistant for components or constructs you use often.

1. To invoke the Language Assistant, select **`Tools`** → **`Language Assistant`** from the HDL Editor pulldown menu.
2. The Language Assistant is divided into three sections: Language Templates, Synthesis Templates, and User Templates. To expand the view of any of these sections, click the '+' next to the topic. Click any of the listed templates to view the HDL code in the right hand pane.
3. Notice the template called D Flip Flop, located under the Flip Flops heading within the Synthesis Templates and the corresponding HDL code in the right-hand window.

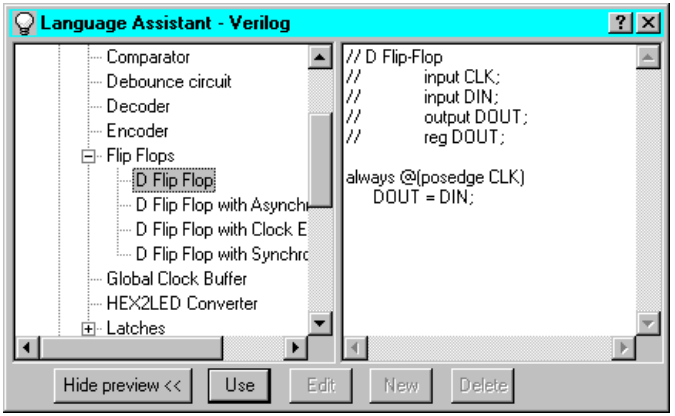


Figure 3-11 HDL Language Assistant

4. Close the Language Assistant by clicking the “X” in the top right corner of its window.
5. Close the HDL editor (do not make any modifications) by selecting **`File`** → **`Exit`**.

Design Description

The HDL design used in this tutorial is either a top-level Verilog or VHDL design file. To ensure that the file has recently been analyzed by Foundation Express, choose **`Synthesis`** → **`Analyze all Sources`**.

HDL files that make up a project always have one of four status indicators associated with the file. Examples of these indicators are shown below:

- A red question mark means that the file has been modified and needs to be re-analyzed. Right-click the file and select Analyze from the pulldown menu.

- A red exclamation point means warnings have been issued. Select the file and examine the warnings under the HDL Warnings tab. To investigate the warning on a file, perform the following steps:

  a) Click the file name

  b) Click the HDL Warnings tab in the messages window

  c) Single-click to highlight the warning message

  d) Press the "F1" key on your keyboard. An extended help message screen will pop up. Many synthesis warnings may be safely ignored.

- A red X means errors have been found.

- A green check mark means that the file is up-to-date with no errors or warnings.

Synthesis

Now that the design has been entered and each file analyzed, you will synthesize the design in the next steps. "Logic Synthesis" is the process of compiling your HDL code into an XNF or EDIF netlist of gates. That netlist is the input to the Xilinx Implementation Tools. In this section, you will "pull" the design completely through synthesis.

1. Set the global synthesis options by selecting **`Synthesis`** → **`Options`**. Set the Default Frequency to 100 (default is 50 MHz), and check the Export Timing Constraints box. Click **`OK`** to accept these values.
2. Click the + next to jct_ver.v or jct_vhd.vhd. This shows the modules within the HDL file.
3. Select the module named "JCT_VER" or "JCT_VHD" and click the Synthesis button from the Flow tab.

This step causes the flow to proceed through Synthesis.

4. In the Synthesis / Implementation window, complete the Target Device fields with this information:
   - Family: SPARTANXL
   - Device: S05XLPC84
   - Speed Grade: -5

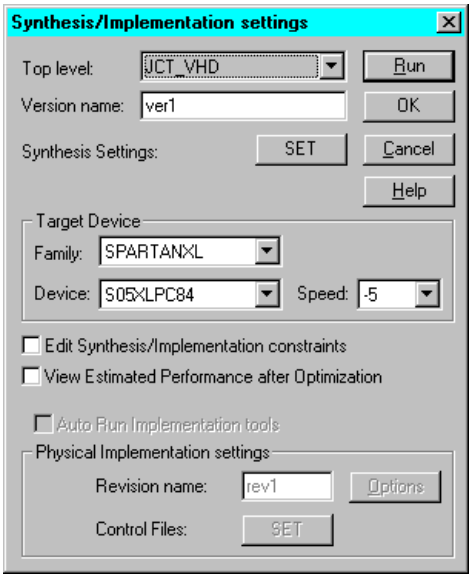


Figure 3-12 Synthesis/Implementation Window

Note: If your design is JCT_VHD, then “JCT_VHD” displays in the Top level field instead of “JCT_VER”.

5. Click **Run**.

   Foundation Express synthesizes and optimizes the design and also creates a design version.

6. Proceed to the next section, “Functional Simulation”.

Functional Simulation

You can perform functional simulation before design implementation to verify that the logic that you have created is correct. Foundation provides a Logic Simulator, which is a gate-level simulator. You can perform functional simulation on a schematic-based design immediately after you have finished using the Schematic editor. In a later section, you will perform timing simulation, which takes place after the design is implemented (placed and routed) with the Xilinx Implementation Tools.

Starting the Logic Simulator

Click the Simulation phase button in the Project Flowchart.

You may be prompted to update the schematic netlist if you modified the design but did not write out a netlist. In this case, click **Yes** to update the netlist. (If you also get a dialog asking to update macros, select **No**. You will get this dialog only if you inadvertently changed something in the HDL editor.)

The Logic Simulator is invoked, and the project netlist is automatically loaded into the simulator.

Performing Simulation

There are three basic steps to simulate your design:

1. Adding signals - the inputs and outputs you want to see in the simulator
2. Adding stimulus - inputs on some of the signals
3. Running the simulation - and view output waveforms

Adding Signals

In order to view signals during the simulation, you must first add them to the Waveform Viewer in the Simulator. The signals are then listed in the Waveform Viewer. You can view and monitor the waveforms next to the corresponding signal names.

Adding Signals Using the Component Selection Window

Follow these steps to add signals using the Component Selection window within the Simulator.

1. Click the Component Selection icon in the toolbar in the Simulator or select **`Signal`** → **`Add Signals`**.

   The Component Selection window opens.

   This window is divided into three panes. The left-most pane is the Signals Selection pane. This pane displays a list of all of the available *signals* for a given level of hierarchy. The middle pane, Chip Selection, displays a list of all of the *components* for a given level of hierarchy.

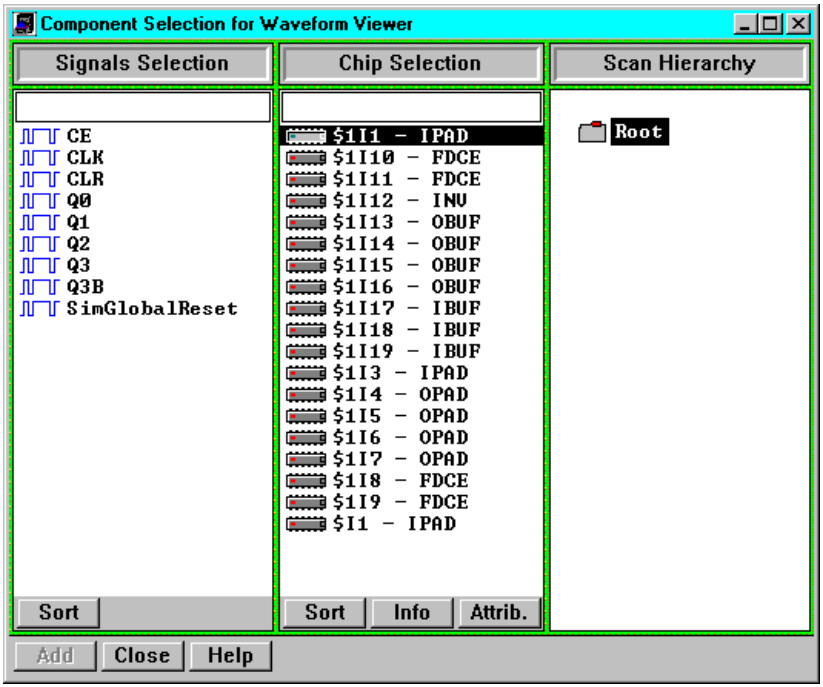


Figure 3-13 Scan Hierarchy Signals Selector (Schematic Design)

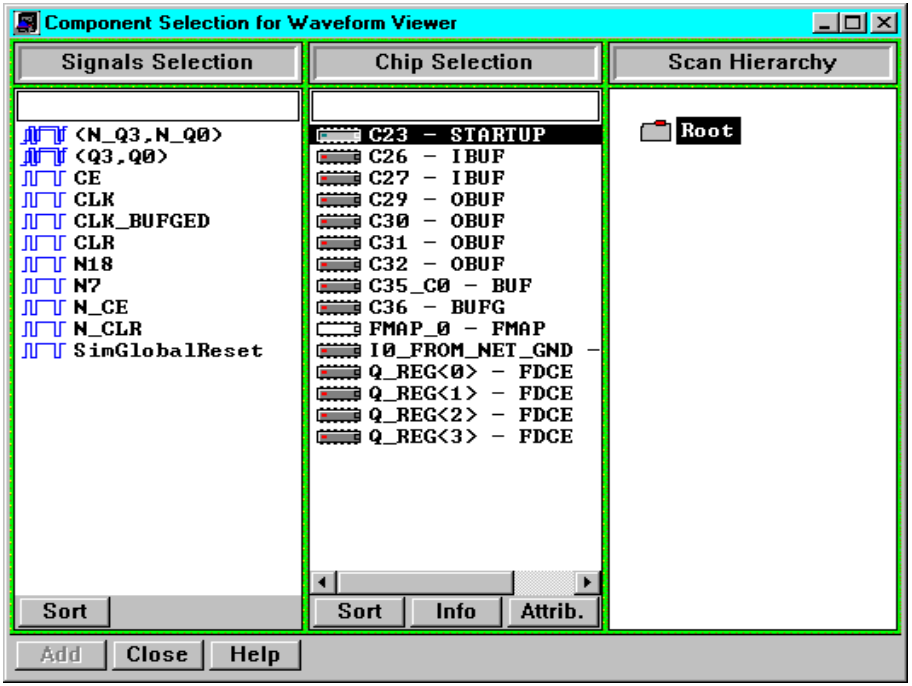


Figure 3-14 Scan Hierarchy Signals Selector (HDL Design)

2. From the Signals Selection pane, you can either double click signals to add them to the Waveform Viewer, or you can single click and then press **Add**. Use whichever method you prefer to add the following nets and buses:

   CLK

   CE

   CLR

   <Q3, Q0> (4-bit wide Q bus)

3. (Schematic design only) Create a bus for the output signals Q3, Q2, Q1, and Q0.

   a) Shift click each Q output.

   b) With all four outputs selected, right-click the mouse and select **Bus** → **Combine**.

   (The HDL design Q outputs are already defined as a bus).

4. Change the radix of the Q bus to binary.
   a) Select Q.
   b) Right-click the mouse button and select **Bus** → **Display Binary**.
5. Close the Component Selection Window by clicking the **Close** button.

All of the signals you added are now in the Waveform Viewer.

Deleting a Signal

To delete any of the signals from the Waveform Viewer, first select the signal in the signal list in the Waveform Viewer, right-click, and then select **Delete Signals** → **Selected**. This operation removes the highlighted signal from the Waveform Viewer.

Adding Stimulus

To define the function of the input signals, you must add stimulus to your simulation. There are many ways to define stimulus with the Foundation Simulator. In this tutorial, you will use the keyboard stimulus.

Open the Stimulator Selection Window by clicking the Stimulator icon in the toolbar or by selecting **Signal** → **Add Stimulators...**

The Stimulator Selection window appears:

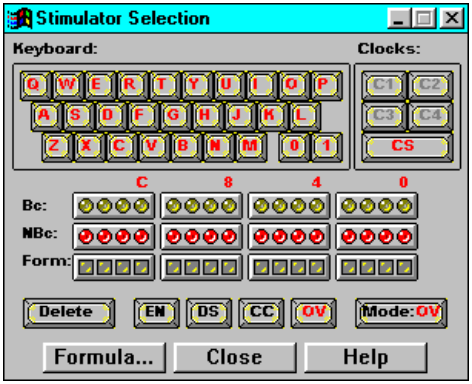


Figure 3-15 Stimulator Selector

Stimulating with the Internal Binary Counter

You need to simulate an input clock for the JCOUNT design, so you will use the Foundation Simulator's internal 16-bit binary counter. Use the least-significant bit (LSB) of the counter (B0). The right-most round yellow LED in the Stimulator Selection window is the B0 counter.

To simulate the system clock, you assign the B0 stimulus to the CLK signal in the simulator.

1. In the Waveform Viewer, select the **CLK** signal by clicking it.
2. In the Stimulator Selection Window, click the B0 stimulator (the right-most yellow round LED). You should now see a B0 next to the CLK signal in the Waveform Viewer indicating that the B0 stimulator is assigned to CLK.
3. Select **Options** → **Preferences** from the Simulator window. This opens the Preferences window. In the Simulation tab of this window, you can set the frequency of the B0 counter output. Set the B0 frequency to 10MHz.

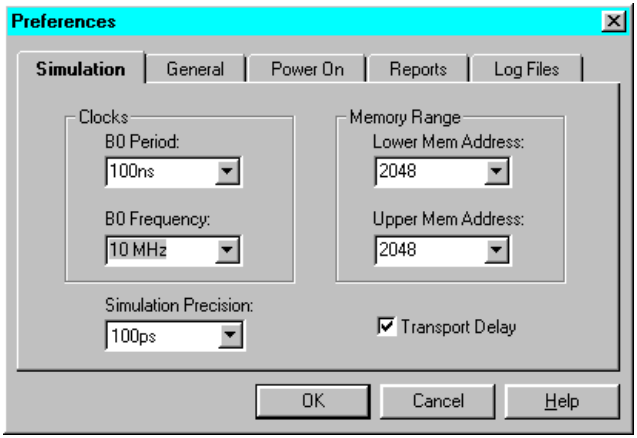


Figure 3-16 Simulator Preferences

4. Click **OK** to close the Preferences window.

Stimulating with Keyboard Stimulators

You assign keyboard keys as stimulus for signals in your design with the keyboard in the Stimulator Selection window. After you assign a key as a stimulus, the signal's value toggles between 1 and 0 whenever you press the corresponding key on your PC's keyboard.

Assign the **R** keyboard stimulus to the CLR signal in the JCOUNT design.

1. Click and drag the **R** key on the keyboard in the Stimulator Selector onto the CLR signal name in the Waveform Viewer.
2. You should now see an **r** next to the CLR signal in the Waveform Viewer, which indicates that this is the assigned stimulus.
3. Press the **R** key on your PC keyboard a few times to see the state of the stimulus toggling in the Waveform Viewer.
4. In the same way, assign the **E** keyboard stimulus to the CE signal in the JCOUNT design.
5. Close the Stimulator Selection window by clicking its Close button.

Running the Simulation

Now you should see the three inputs of the JCOUNT design, CLK, CLR, and CE, listed in the Waveform Viewer, each having some type of stimulus associated with it. You should also see the Q output listed. You are now ready to run the simulation.

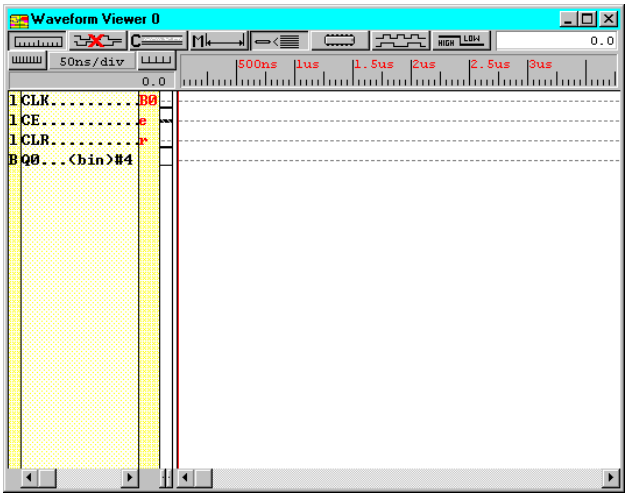


Figure 3-17 Signals with Stimulus

Use the Step button in the Simulator toolbar to advance the simulation for a set amount of time. You can define the size of the Step using the pulldown menu next to the Step button, shown in the following diagram.

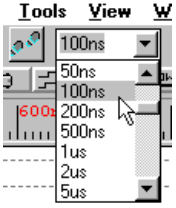


Figure 3-18 Simulator Step

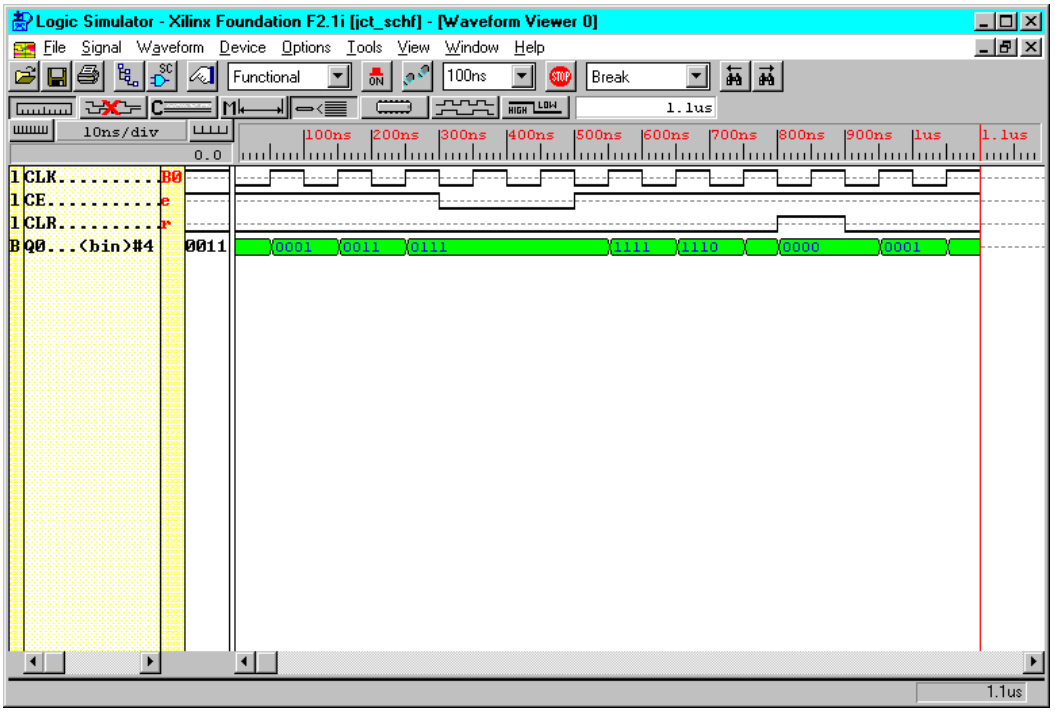
1. Set the Step size to 100ns.
2. Press the **R** key on your PC keyboard until the CLR stimulus state is low.

3. Press the **E** key on your PC keyboard so that the CE state is high.
4. Click the **Step** button three times to advance the simulation.

The CLK signal is clocking based on the B0 frequency you set earlier.

5. Press the **E** key on your PC keyboard to simulate the clock enable signal.
6. Click the **Step** button twice to advance the simulation.
7. Press the **E** key again on your PC keyboard to simulate.
8. Click the **Step** button three times to advance the simulation.
9. Press the **R** key on the PC keyboard to set CLR to high.
10. Click the Step button once.
11. Press the **R** key on the PC keyboard to set CLR to low.
12. Click the Step button twice.

Does the circuit appear to be working properly? Is the counter counting? The waveform should look like the following figure.



13. To clear your waveforms and start your simulation over, choose **Waveform** → **Delete** → **All Waveforms with Power On** from the Simulator pulldown menus.
14. Run several simulations, experimenting with the CLR and CE input functionality.
15. Close the Logic Simulator window by selecting **File** → **Exit**.

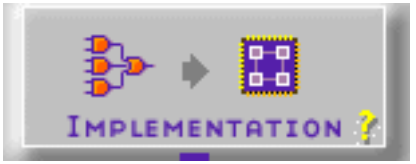
Implementation

This section describes how to implement a design.

- If you created the schematic design, proceed to the "Implementing the Schematic Design" section.
- If you created the HDL design, proceed to the "Implementing the HDL Design" section.

Implementing the Schematic Design

To begin the implementation of the schematic design, click the implementation phase button in the Project Flow diagram.



If you are asked if you wish to update the EDIF netlist because the schematic is newer, say Yes to update the EDIF netlist. This EDIF netlist is the actual input file to the Design Implementation tools.

Next you will see the Implement Design dialog box.

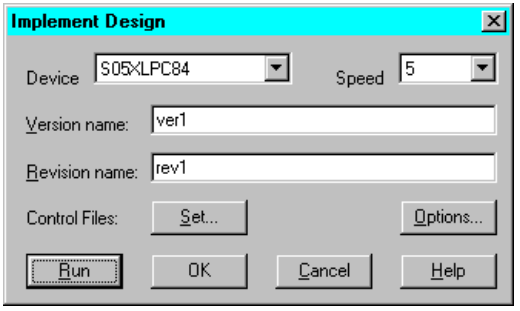


Figure 3-19 Implement Design Dialog Box

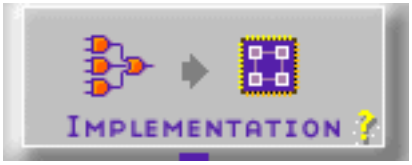
With this dialog box, you can select the target device and various implementation options. The target device is already set to S05XLPC84-5 because that was the device selected when the Foundation project was created. The Version and Revision fields have been filled in automatically. You can also find these version and revision names in the Project Manager Versions tab after implementation.

Proceed to the “Implementation Options” section.

Implementing the HDL Design

In the “Synthesis” section, you synthesized your design. To implement the design, perform the following steps.

1. Click the Implementation phase button in the Project Flow diagram.



2. The Revision Name is automatically filled in with “rev1”. If you want to use a new name, enter it in the box. Proceed to the “Implementation Options” section.

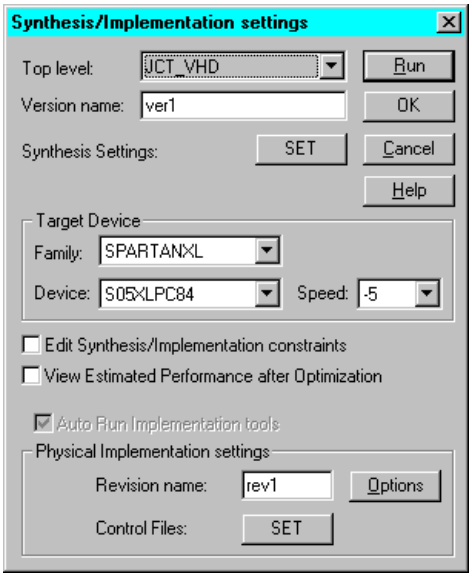


Figure 3-20 Synthesis/Implementation Dialog Box

Implementation Options

Implementation options specify how a design is optimized, mapped, placed, routed, and configured. Options are grouped into objects called implementation, simulation, and configuration templates. Each template defines an implementation, simulation or configuration approach.

1. Click the **`Options`** button. The Options dialog box opens.

2. Select desired options and click **OK**.

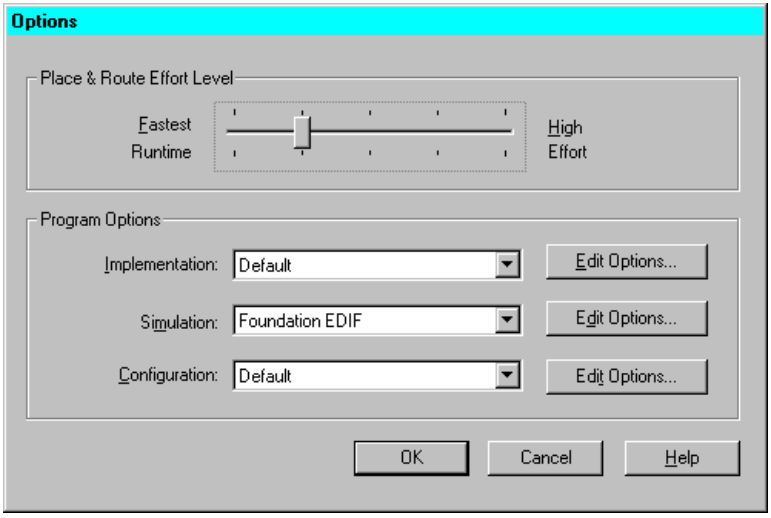


Figure 3-21 Options Dialog Box

3. Click **OK** to close the Options dialog box.

Running Implementation — The Flow Engine

After setting the implementation options, you are ready to implement the design.

Click **Run** in the Implement Design dialog box or click **Run** in the Synthesis/Implementation dialog box.

The Flow Engine displays and processes your design through the implementation steps.

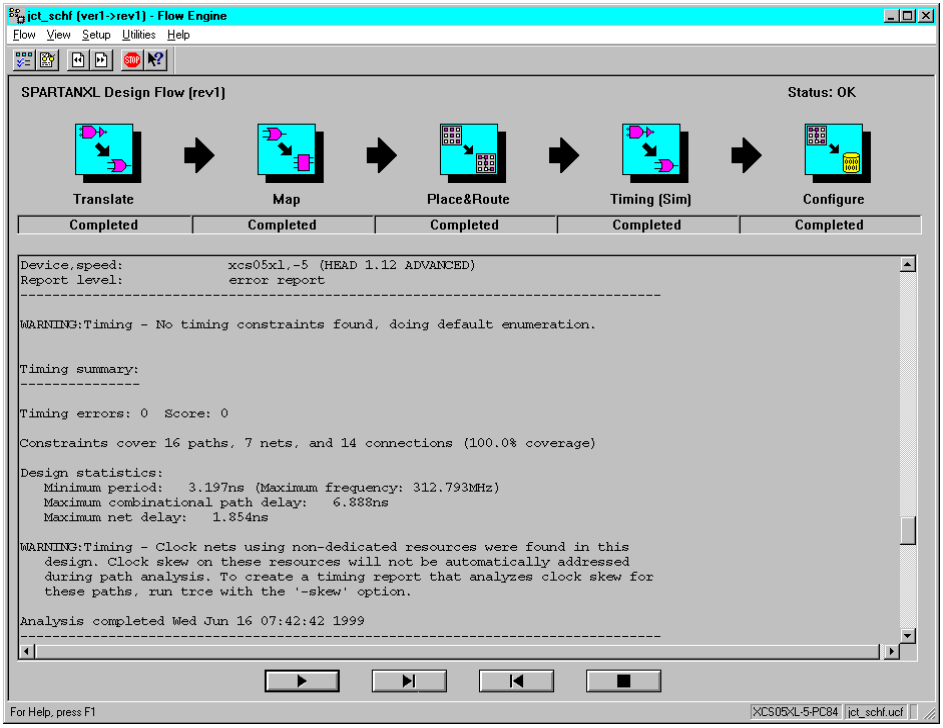


Figure 3-22 Flow Engine

When implementation is complete, the Flow Engine closes automatically, and the Foundation Series Project Manager is fully visible again. Click **OK** in the Flow Engine Completed Successfully dialog box.

The status of the implementation is displayed in the console window at the bottom of the Project Manager. You should see (OK Implemented) and Completed Successfully for the version and revision. If you encountered any errors in the implementation, refer to the Implementation Log file for details on the error.

Viewing Implementation Results

The Foundation Series Project Manager maintains control over all of your design implementation versions and revisions. You can directly view and analyze these implementations from the Project Manager.

1. Click the Versions tab in the left-hand pane of the Project Manager. You should see a hierarchical display of the implementation you just ran. The revision that is most current is displayed in bold.

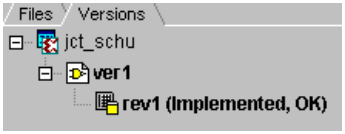


Figure 3-23 Versions Tab (Schematic Design)

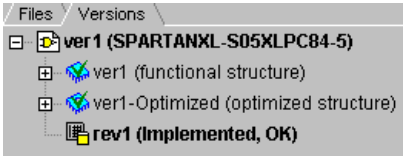


Figure 3-24 Versions Tab (HDL Design)

2. With the current revision selected, click the Reports tab in the right-hand side of the Project Manager. The Reports tab displays reports and logs for the selected revision of the design.
3. Double click the report entitled Implementation Report Files. This displays the Xilinx Report Browser, which contains all of the implementation reports. You have the option to browse through any of these reports at this time.

Timing Simulation

Timing simulation uses the block and routing delay information from the routed design to give a more accurate assessment of the behavior of the circuit under worst-case conditions. For this reason, timing simulation is performed after the design has been placed and routed.

This section of the tutorial shows how to run a script to do a simulation. Timing simulation uses the same tools as does the functional simulation that you did with the design earlier. The only difference is that the design which is loaded into the simulator for timing simulation contains worst-case routing delays based on the actual placed and routed design.

Invoking Timing Simulation

To invoke the timing simulator, click the Timing Simulation icon in the Verification phase button in the Project Manager Flow diagram.

The simulator is now loaded and ready to simulate. For this simulation, you will use an existing script file.

Simulating with Script Files - Script Editor

Earlier, you used the Foundation Series functional simulator to input signals to the simulator and to provide stimulus including keyboard stimulus and the internal binary counter. In this section, you use a script file to simulate the design and will launch it from the Script Editor.

1. To invoke the Script Editor, select **`Tools`** → **`Script Editor`** from the pulldown menus within the Simulator. A dialog box prompts you to select a script file.
2. Choose **`Open: Existing Script File`** and click **`OK`**.
3. Select the file: `jcount.cmd`, which was written previously for you to use in this tutorial.

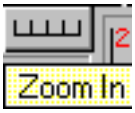
Browse the script file; you will notice the same inputs and outputs that you used in the functional simulation earlier.

Running the Simulation from the Script Editor

1. You can execute the simulation directly from the Script Editor. To do this, select **`Execute`** → **`Go`**.

   A log of the executed commands appears at the bottom of the Script Editor.
2. To view the simulation results in the Waveform Viewer, move the Script Editor window and bring the Waveform Viewer window to the front of your view.

3. Click the Expand icon located above the signal names display to view the waveform in more details.



4. Inspect the simulation results to make sure they are accurate.

You should now see that this is indeed performing a timing simulation based on actual delays in the placed and routed design. If you zoom in to get a closer view of the waveforms, you will see that there is a delay from the rising edge of the clock to the transitions or the counter outputs.

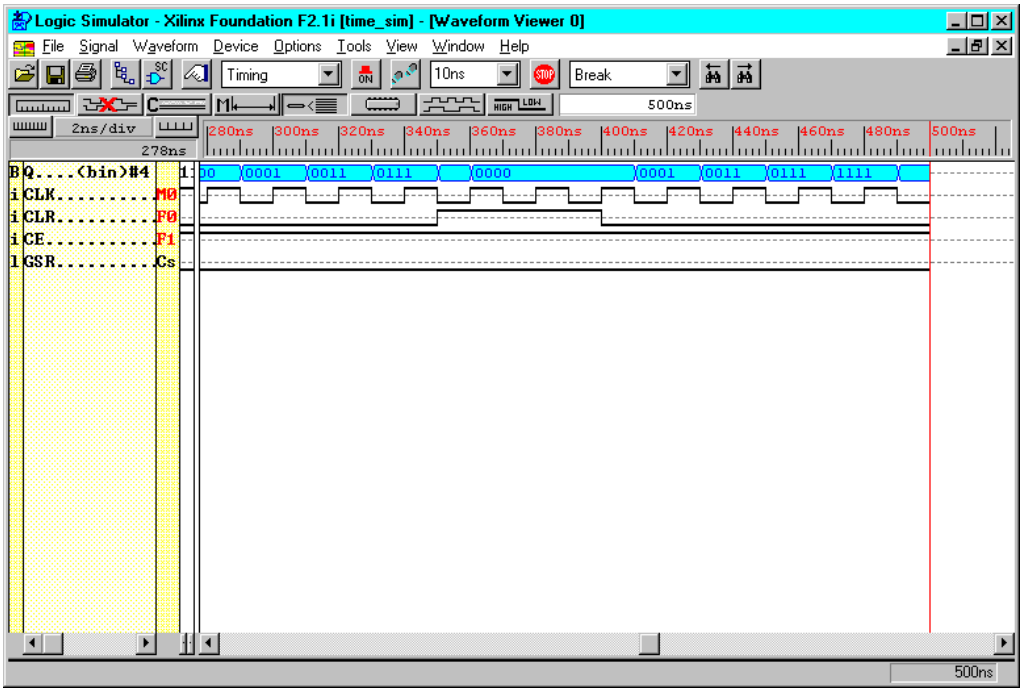


Figure 3-25 Timing Simulation Waveforms

Closing the Simulator

When you are satisfied with the results of the simulation, you may close the Script Editor and the Simulator.

Appendix A

Glossary

This appendix contains definitions and explanations for terms used in the *Foundation Series 2.1i Quick Start Guide.*

ABEL

ABEL is a high-level language (HDL) and compilation system produced by Data I/O Corporation.

actions

In state machines, actions are HDL statements that are used to make assignments to output ports or internal signals. Actions can be executed at several points in a state diagram. The most commonly used actions are state actions and transition actions. State actions are executed when the machine is in the associated state. Transition actions are executed when the machine goes through the associated transition.

Aldec

An Electronic Design Automation (EDA) vendor. Aldec provides the Foundation Project Manager, Schematic Editor, Logic Simulator, and HDL Editor, and State Editor.

aliases

Aliases, or signal groups, are useful for probing specific groups of nodes.

analyze

The Foundation Express process in which design source files are examined for correct syntax.

architecture

Architecture is the common logic structure of a family of programmable integrated circuits. The same architecture can be realized in different manufacturing processes. Examples of Xilinx architectures are the XC4000, XC5200, and XC9500 devices.

attributes

Attributes are instructions placed on symbols or nets in an FPGA schematic to indicate their placement, implementation, naming, direction, or other properties.

back-annotation

Back-annotation is the translation of a routed or fitted design to a timing simulation netlist.

BitGen

A program that produces a bitstream for Xilinx device configuration. BitGen takes a fully routed NCD (Circuit Description) file as its input and produces a configuration bitstream, a binary file with a .bit extension.

Black Box Instantiation

Instantiation where the synthesizer is not given the architecture or modules. In Foundation, black boxes are translated with the implementation tools.

block

1. A block is a group of one or more logic functions.
2. A block is a schematic or symbol sheet. There are four types of blocks.
   - A Composite block indicates that the design is hierarchical.
   - A Module block is a symbol with no underlying schematic.
   - A Pin block represents a schematic pin.
   - An Annotate block is a symbol without electrical connectivity that is used only for documentation and graphics.

breakpoint

A breakpoint is a condition for which a simulator must stop to perform simulation commands.

buffer

A buffer is an element used to increase the current or drive of a weak signal and, consequently, increase the fanout of the signal. A storage element.

bus

A bus is a group of nets carrying common information. In LogiBLOX, bus sizes are declared so that they can be expanded accordingly during design implementation.

CLB

The Configurable Logic Block (CLB). Constitutes the basic FPGA cell. It includes two 16-bit function generators (F or G), one 8-bit function generator (H), two registers (flip-flops or latches), and reprogrammable routing controls (multiplexers).

CLBs are used to implement macros and other designed functions. They provide the physical support for an implemented and downloaded design. CLBs have inputs on each side, and this versatility makes them flexible for the mapping and partitioning of logic.

component

A component is an instantiation or symbol reference from a library of logic elements that can be placed on a schematic.

condition

A condition is a Boolean expression. If there is more than one transition leaving a state in a state machine, you must associate a condition with each transition.

constraint

Constraints are specifications for the implementation process. There are several categories of constraints: routing, timing, area, mapping, and placement constraints.

Using attributes, you can force the placement of logic (macros) in CLBs, the location of CLBs on the chip, and the maximum delay between flip-flops. CLBs are arranged in columns and rows on the FPGA device. The goal is to place logic in columns on the device to attain the best possible placement from the standpoint of both performance and space.

constraints editor

A GUI tool that you can use to enter design constraints. In Foundation 2.1, there are two constraint editors. The Express editor is available only in the Foundation Express product configuration. The Xilinx Constraints Editor is integrated with the Design Implementation tools and available in all product configurations.

constraint file

A constraint file specifies constraints (location and path delay) information in a textual form. An alternate method is to place constraints on a schematic.

CORE Generator tool

A software tool for generating and delivering parameterizable cores optimized for FPGAs. The library includes cores as complex as DSP filters and multipliers and as simple as delay elements. You can use these cores as building blocks in order to complete your designs more quickly.

CPLD

Complex Programmable Logic Device (CPLD) is an erasable programmable logic device that can be programmed with a schematic or a behavioral design. CPLDs constitute a type of complex PLD based on EPROM or EEPROM technology. They are characterized by an architecture offering high speed, predictable timing, and simple software.

The basic CPLD cell is called a macrocell, which is the CPLD implementation of a CLB. It is composed of AND gate arrays and is surrounded by the interconnect area.

CPLDs consume more power than FPGA devices, are based on a different architecture, and are primarily used to support behavioral designs and to implement complex counters, complex state machines, arithmetic operations, wide inputs, and PAL crunchers.

CPLD fitter

The CPLD Fitter implements designs for the XC9500 devices.

design entry tools

A set of tools accessible from the Project Manager. These tools include the Schematic Editor, State Editor, and HDL Editor.

Foundation Express, an embedded portion of the Foundation software package, contains the VHDL and Verilog design languages.

daisy chain

A daisy chain is a series of bitstream files concatenated in one file. It can be used to program several FPGAs connected in a daisy chain board configuration.

design implementation tools

A set of tools that comprise the mainstream programs offered in the Xilinx design implementation tools. The tools are NGDBuild, MAP, PAR, NGDAnno, TRCE, all the NGD2 translator tools, BitGen, and PROMGen. The GUI-based tools are Flow Engine, Constraint Editor, FPGA Editor, Floorplanner, PROM File Formatter, JTAG Programmer, and Hardware Debugger.

effort level

Effort level refers to how hard the Xilinx Design System (XDS) tries to place a design. The effort level settings are.

- High, which provides the highest quality placement but requires the longest execution time. Use high effort on designs that do not route or do not meet your performance requirements.
- Medium, which is the default effort level. It provides the best trade-off between execution time and high quality placement for most designs.
- Low, which provides the fastest execution time and adequate placement results for prototyping of simple, easy-to-route designs. Low effort is useful if you are exploring a large design space and only need estimates of final performance.

elaborate

The HDL process that combines the individual parts of a design into a single design and then synthesizes the design.

EXORmacs (Motorola)

EXORmacs is a PROM format supported by the Xilinx tools. Its maximum address is 16 777 216. This format supports PROM files of up to (8 x 16 777 216) = 134 217 728 bits.

fanout

Fanout is the maximum number of specified unit loads that a specified output can drive.

fitter

The fitter is the software that maps a PLD logic description into the target CPLD.

floorplanning

Floorplanning is the process of choosing the best grouping and connectivity of logic in a design.

It is also the process of manually placing blocks of logic in an FPGA where the goal is to increase density, routability, or performance.

FPGA

Field Programmable Gate Array (FPGA) is a class of integrated circuits pioneered by Xilinx in which the logic function is defined by the customer using Xilinx development system software after the IC has been manufactured and delivered to the end user. Gate arrays are another type of IC whose logic is defined during the manufacturing process. Xilinx supplies RAM-based FPGA devices.

FPGA applications include fast counters, fast pipelined designs, register intensive designs, and battery powered multi-level logic.

functional simulation

Functional simulation is the process of identifying logic errors in your design before it is implemented in a Xilinx device. Because timing information for the design is not available, the simulator tests the logic in the design using unit delays. Functional simulation is usually performed at the early stages of the design process.

gate

A gate is an integrated circuit composed of several transistors and capable of representing any primitive logic state, such as AND, OR, XOR, or NOT inversion conditions. Gates are also called digital, switching, or logic circuits.

guided design

Guided design is the use of a previously implemented version of a file for design mapping, placement, and routing. Guided design allows logic to be modified or added to a design while preserving the layout and performance that have been previously achieved.

guided mapping

An existing NCD file is used to "guide" the current MAP run. The guide file may be used at any stage of implementation: unplaced or placed, unrouted or routed. In 2.1i, guided mapping is supported through the Project Manager.

HDL

Hardware Description Language. A language that describes circuits in textual code. The two most widely accepted HDLs are VHDL and Verilog.

An HDL, or hardware description language, describes designs in a technology-independent manner using a high level of abstraction.

HDL Editor

Foundation's editor for ABEL, Verilog, and VHDL. The HDL Editor also provides a syntax checker, language templates, and access to the synthesis tools.

hierarchical design

A hierarchical design is a design composed of multiple sheets at different levels of your schematic.

Hierarchy Browser

The left-hand portion of the Foundation Project Manager that displays the current design project. The browser also displays two tabs, Files and Versions.

implementation

Implementation is the mapping, placement, and routing of a design. A phase in the design process during which the design is placed and routed. (For CPLDs, the design is fitted.)

instantiation

Incorporating a macro or module into a top-level design. The instantiated module can be a LogiBLOX module, CORE-generated module, VHDL module, Verilog module, schematic module, state machine, or netlist.

Language Assistant

The Language Assistant in the HDL Editor provides templates to aid you in common VHDL and Verilog constructs, common logic functions, and architecture-specific features.

Library Manager

The Library Manager is used to perform a variety of operations on the design entry tools libraries and their contents. These libraries contain the primitives and macros that you use to build your design.

locking

Lock placement applies a constraint to all placed components in your design. This option specifies that placed components cannot be unplaced, moved, or deleted.

LogiBLOX

Xilinx design tool for creating high-level modules such as counters, shift registers, and multiplexers.

logic

Logic is one of the three major classes of ICs in most digital electronic systems: microprocessors, memory, and logic. Logic is used for data manipulation and control functions that require higher speed than a microprocessor can provide.

Logic Simulator

The Logic Simulator, a real-time interactive design tool, can be used for both functional and timing simulation of designs. The Logic Simulator creates an electronic breadboard of your design directly from your design's netlist. The Logic Simulator can be accessed by clicking the Functional Simulation icon on the Simulation phase button or the Timing Simulation icon on the Verification phase button in the Project Manager.

macro

A macro is a component made of nets and primitives, flip-flops or latches, that implements high-level functions, such as adders, subtractors, and dividers. Soft macros and RPMs are types of macros.

A macro can be unplaced, partially placed, or fully placed; it can also be unrouted, partially routed, or fully routed. See also "physical macro."

mapping

Mapping is the process of assigning a design's logic elements to the specific physical elements that actually implement logic functions in a device.

MCS file

An MCS file is an output from the PROMGen program in Intel's MCS-86 format.

MRP file

An MRP (mapping report) file is an output of the MAP run. It is an ASCII file containing information about the MAP run. The information in this file contains DRC warnings and messages, mapper warnings and messages, design information, schematic attributes, removed logic, expanded logic, signal cross references, symbol cross references, physical design errors and warnings, and a design summary.

NCD file

An NCD (netlist circuit description) file is the output design file from the MAP program, LCA2NCD, PAR, or FPGA Editor. It is a flat physical design database correlated to the physical side of the NGD in order to provide coupling back to the user's original design. The NCD file is an input file to MAP, PAR, TRCE, BitGen, and NGDAnno.

net

A net is a logical connection between two or more symbol instance pins. After routing, the abstract concept of a net is transformed to a physical connection called a wire.

A net is an electrical connection between components or nets. It can also be a connection from a single component. It is the same as a wire or a signal.

netlist

A netlist is a text description of the circuit connectivity. It is basically a list of connectors, a list of instances, and, for each instance, a list of the signals connected to the instance terminals. In addition, the netlist contains attribute information.

NGA file

An NGA (native generic annotated) file is an output from the NGDAnno run. An NGA file is subsequently input to the appropriate NGD2 translation program.

NGDAnno

The NGDAnno program distributes delays, setup and hold time, and pulse widths found in the physical NCD design file back to the logical NGD file. NGDAnno merges mapping information from the NGM file and timing information from the NCD file and puts all this data in the NGA file.

NGDBuild

The NGDBuild program performs all the steps necessary to read a netlist file in XNF or EDIF format and create an NGD file describing the logical design. The GUI equivalent is called Translate.

NGD file

An NGD (native generic database) file is an output from the NGDBuild run. An NGD file contains a logical description of the design expressed both in terms of the hierarchy used when the design was first created and in terms of lower-level Xilinx primitives to which the hierarchy resolves.

NGM file

An NGM (native generic mapping) file is an output from the MAP run and contains mapping information for the design. The NGM file is an input file to the NGDAnno program.

one-hot encoding

For state machines, in one-hot encoding, an individual state register is dedicated to one state. Only one flip-flop is active, or hot, at any one time.

optimization

Optimization is the process that decreases the area or increases the speed of a design. Foundation allows you to control optimization of a design on a module-by-module basis. This means that you have the ability to, for instance, optimize certain modules of your design for speed, some for area, and some for a balance of both.

optimize

The third step in the FPGA Express synthesis flow. In this stage, the implemented design is re-synthesized with constraints the user specifies. This is the final step before writing out the XNF file from FPGA Express.

PAR (Place and Route)

PAR is a program that takes an NCD file, places and routes the design, and outputs another NCD file. The NCD file produced by PAR can be used as a guide file for reiterative placement and routing. The NCD file can also be used by the bitstream generator, BitGen.

path delay

A path delay is the time it takes for a signal to propagate through a path.

PCF file

The PCF file is the “Physical Constraints File” created by the MAP program. It is an ASCII file containing physical constraints created by the MAP program as well as physical constraints you enter. You can edit the PCF file from within the FPGA Editor.

PDF file

Project Description File. The PDF file contains library and other project-specific information. Not to be confused with an Adobe Acrobat document with the same extension.

physical Design Rule Check (DRC)

Physical Design Rule Check (DRC) is a series of tests to discover logical and physical errors in the design. Physical DRC is applied from FPGA Editor, BitGen, PAR, and Hardware Debugger. By default, results of the DRC are written into the current working directory.

pin

A pin can be a symbol pin or a package pin. A package pin is a physical connector on an integrated circuit package that carries signals into and out of an integrated circuit. A symbol pin, also referred to as an instance pin, is the connection point of an instance to a net.

pinwires

Pinwires are wires which are directly tied to the pin of a site (that is, CLB, IOB).

PLD

A PLD, or programmable logic device, is composed of two types of gate arrays: the AND array and the OR array, thus providing for sum of products algorithmic representations. PLDs include three distinct types of chips: PROMs, PALs, and PLAs. The most flexible device is the PLA (programmable logic array) in which both the AND and OR gate arrays are programmable. In the PROM device, only the OR gate array is programmable. In the PAL device, only the AND gate array is programmable. PLDs are programmed by blowing the fuses along the paths that must be disconnected.

FPGAs and CPLDs are classes of PLDs.

Project Manager

The primary GUI for managing a Foundation Project. Design entry, synthesis, simulation, implementation, and downloading can be launched from the Project Manager.

Project Flowchart

The right-hand portion of the Foundation Project Manager that provides access to the synthesis and implementation tools, and the current design project. The Project Flowchart can display up to four tabs: Flow, Contents, Reports, and Synthesis.

PROM File Formatter

The PROM File Formatter is the program used to format one or more bitstreams into an MC86, TEKHEX, EXORmacs or HEX PROM file format.

radix

A radix is the base—usually binary, octal, decimal, or hexadecimal—in which waveforms are displayed in a waveform viewer.

readback

Readback is the process of reading the logic downloaded to an FPGA device back to the source. There are two types of readbacks:

- A readback of logic usually accompanied by a comparison check to verify that the design was downloaded in its entirety.
- A readback of the states stored in the device memory elements to ensure that the device is behaving as expected.

route

The process of assigning logical nets to physical wire segments in the FPGA that interconnect logic cells.

route-through

A route that can pass through an occupied or an unoccupied CLB site is called a route-through. You can manually do a route-through in the FPGA Editor. Route-throughs provide you with routing resources that would otherwise be unavailable.

Schematic Flow

If a project is defined as a Schematic Flow, no Synthesis button displays in the Project Flow area of the Project Manager. A Schematic Flow may only have schematic designs as top-level designs. However, these top-level designs can contain HDL modules. If the designs contain HDL modules, the Synthesis tab displays in the upper portion of the Project Flow area.

states

The values stored in the memory elements of a device (flip-flops, RAMs, CLB outputs, and IOBs) that represent the state of that device for a particular readback (time). To each state, there corresponds a specific set of logical values.

state diagram

A state diagram is a pictorial description of the outputs and required inputs for each state transition as well as the sequencing between states. Each circle in a state diagram contains the name of a state. Arrows to and from the circles show the transitions between states and the input conditions that cause state transitions. These conditions are written next to each arrow.

state machine

A state machine is a set of combinatorial and sequential logic elements arranged to operate in a predefined sequence in response to specified inputs. The hardware implementation of a state machine design is a set of storage registers (flip-flops) and combinatorial logic, or gates. The storage registers store the current state, and the logic network performs the operations to determine the next state. See also “symbolic state machine” and “encoded state machine.”

state table

A state table shows the value of the outputs for all combinations of current states and inputs. It also defines the next state for each set of inputs.

static timing analysis

A static timing analysis is a point-to-point delay analysis of a design network.

static timing analyzer

A static timing analyzer is a tool that analyzes the timing of the design on the basis of its paths.

status bar

The status bar is an area located at the bottom of a tool window that provides information about the commands that you are about to select or that are being processed.

stimulus information

Stimulus information is the information defined at the schematic level and representing a list of nodes and vectors to be simulated in functional and timing simulation.

synthesis

The HDL design process in which each design module is elaborated and the design hierarchy is created and linked to form a unique design implementation. Synthesis starts from a high level of logic abstraction (typically Verilog or VHDL) and automatically creates a lower level of logic abstraction using a library containing primitives.

TEKHEX (Tektronix)

TEKHEX (Tektronix) is a PROM format supported by Xilinx. Its maximum address is 65 536. This format supports PROM files of up to (8 x 65 536) = 524 288 bits.

TRCE

TRCE (Timing Reporter and Circuit Evaluator) “trace” is a program that will automatically perform a static timing analysis on a design using the specified (either timing constraints. The input to TRCE is an NCD file and, optionally, a PCF file. The output from TRCE is an ASCII timing report which indicates how well the timing constraints for your design have been met.

TWR file

A TWR (Timing Wizard Report) file is an output from the TRCE program. A TWR file contains a logical description of the design expressed both in terms of the hierarchy used when the design was first created and in terms of lower-level Xilinx primitives to which the hierarchy resolves.

UCF file

A UCF (User Constraints File) contains user-specified logical constraints.

verification

Verification is the process of reading back the configuration data of a device and comparing it to the original design to ensure that all of the design was correctly received by the device.

Verilog

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

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

VHDL

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

VHDL is an acronym for VHSIC Hardware Description Language, which can be used to model a digital system at many levels of abstraction ranging from the algorithmic level to the gate level. It is IEEE standard 1076-1993.

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

wire

A wire is either 1) a net or 2) a signal.

Index

D

E

F

R

S

T

U

V

W

X