Foundation Series Quick Start Guide 1.4

Features of Foundation Series 1.4

Setting Up the Foundation Tools

Foundation Overview

In-depth Tutorial

Glossary

Foundation Series Quick Start Guide 1.4 — 0401697 01 *Printed in U.S.A.*

Preface

About this Quick Start Guide

This Quick Start Guide is intended to give an overview of the features and additions to Xilinx's newest product—Foundation 1.4. The primary focus of this guide is to show the relationship between the design entry tools and the design implementation tools.

This guide should be used as the initial learning tool for designers who are unfamiliar with the features of the Foundation software. It will also serve as a good reference for more advanced users who simply need a refresher on a particular subject, such as components or constraints.

Quick Start Guide Contents

This guide covers the following topics:

- **Chapter 1**, "Features of Foundation Series 1.4," introduces the new and enhanced features of the Foundation 1.4 software and provides information about both FPGA- and CPLD-specific features.
- **Chapter 2**, "Setting Up the Foundation Tools," gives instructions for the installation of Foundation 1.4 and provides you with information about the type of computer you need to successfully implement your designs.
- **Chapter 3**, "Foundation Overview," looks in-depth at the capability and flexibility of the Foundation software.

- **Chapter 4**, "In-depth Tutorial," is a tutorial that explains many different facets of a schematic-oriented Foundation design flow. This chapter also shows how to use Foundation accessories such as the State Editor, the Waveform Editor, and the HDL Editor.
- **Appendix A**, "Glossary," defines some of the commonly used terms in this Guide.

Conventions

Typographical

This manual uses the following conventions. An example illustrates each convention.

- `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.

  `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*

  Square brackets also enclose footnotes in tables that are printed out as hardcopy in DynaText®.

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

  `lowpwr ={on|off}`

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

  `symbol` *editor_name* `[bus|pins]`

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

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

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

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

Online Document

Xilinx has created several conventions for use within the DynaText online documents.

- Red-underlined text indicates an interbook link, which is a cross-reference to another book. Click on 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 on the blue-underlined text to open the specified cross-reference.
- There are several types of icons.

  Iconized figures are identified by the figure icon.

Figure 1-1 Naming Conventions 

Iconized tables are identified by the table icon.

Table 13-14 Carry Modes 

The Copyright icon displays in the upper left corner on the first page of every Xilinx online document.

The DynaText footnote icon displays next to the footnoted text.

Macro 

Double-click on these icons to display figures, tables, copyright information, or footnotes in a separate window.

- Inline figures display within the text of a document. You can display these figures in a separate window by clicking on the figure.

Contents

Chapter 1

Features of Foundation Series 1.4

Welcome to Xilinx's newest software release, known as Foundation® Series 1.4. Xilinx is the world's largest supplier of programmable logic solutions, including industry-leading device architectures and world-class design software.

This release contains many enhancements and additions. The result is a product that provides hardware designers with an improved suite of tools for designing and implementing Programmable Logic Devices (PLDs).

If you ordered Foundation Express, then you also received the new Synopsys® HDL design entry tool that can be used to create VHDL and Verilog® designs. Refer to the Foundation Express documentation, *Foundation Express User Guide* and *Foundation Express Application Note Supplement*, for details.

This chapter contains the following sections:

- "Xilinx Device Support" section
- "Foundation Project Manager" section
- "Design Entry Tools" section
- "Design Implementation Tools" section

Note: Foundation Series 1.4 is intended for use with the Xilinx design implementation tools, such as PAR and MAP. Complete compatibility with the XNF (logical) and LCA (physical) design files that were used in previous XACT*step* releases has been preserved.

To see a step-by-step installation procedure, see the "Installing Foundation" chapter in the *Foundation Series 1.4 Install and Release Document*.

Xilinx Device Support

Foundation Series 1.4 supports the XC3000A / L, XC3100A / L, XC4000E / L / EX / XL / XV, XC5200, Spartan, and XC9500 devices.

For further details about device support, refer to the "Device and Package Support" chapter in the *Foundation Series 1.4 Install and Release Document* that you received with this software.

Note: Technical information for all but the latest families is included in the Xilinx *Programmable Logic Data Book*. For the most up-to-date information about new devices, please see the Xilinx Web Site at http: / / www.xilinx.com

Foundation Project Manager

The Foundation Project Manager—the overall project management tool—contains the Foundation Series tools used in the design process. Within the Project Manager, you access both the design entry tools and the design implementation tools. Major new Project Manager features include the following:

- new tab views in the Project Flowchart area (Flow, Contents, Status, Reports, Synthesis). For a description of the new tabs, see the "Project Manager Tabs" section of the *Foundation Series User Guide*.
- improved performance of the netlist converter
- enhanced XNF support
  - expanded support for third-party vendors
  - importing of hierarchical XNF
- long macro names support (127 characters)
- enhanced project and library archiving (Refer to the "Project Archiving" section of the *Foundation Series User Guid*e.)

Design Entry Tools

This section describes both the new and previously available features of the design entry tools. You can also find a list of the new design entry features in the online help. From the Project Manager, select **Help** → **Foundation Help Contents** → **What's New**.

Major new features that apply to the design entry tools suite include the following:

- year 2000 compliant
- larger design support
- longer name support (127 characters) for design elements (macros, nets, buses, pins, and reference designators)
- new file formats (Designs created prior to Foundation Series 1.4 are backed up before conversion.)

EDIF Netlist Format Supported

The Project Manager design entry tools of Foundation Series 1.4 generate EDIF, which is the preferred netlist file format. (Foundation Express generates XNF files.)

Note: All designs must be created using components from the Xilinx Unified Libraries, described fully in Xilinx's online document, the *Libraries Guide*.

LogiBLOX

In Foundation Series 1.4, the LogiBLOX interface is integrated into the Schematic Editor. Within the Schematic Editor, select **Options** → **LogiBLOX** or **Options** → **Import LogiBLOX**. The Import LogiBLOX option allows you to import a LogiBLOX component that has been previously created in another design project into the current project.

LogiBLOX can be used to create the following types of modules, among others:

- ROMs
- RAMs
- counters
- comparators
- decoders

LogiBLOX modules can be used in both schematic and HDL-based designs. Refer to the "LogiBLOX" chapter in the *Foundation Series User Guide* for details.

XVHDL Compiler

XVHDL compiler features include the following:

- uses EDIF format for output netlist
- supports hierarchy in netlist
- infers LogiBLOX components
- produces combinatorial logic usage estimate
- contains an attribute to control clock buffer assignment
- maintains net names that correspond closely to the signals defined in the VHDL source

For specific information about how to create VHDL designs and use the XVHDL compiler, refer to the "Top-level VHDL Designs" section in the *Foundation Series User Guide.*

XABEL Compiler

XABEL offers the following features:

- uses EDIF format for output netlist
- supports hierarchical design source modules
- supports several Xilinx attributes that help control design implementation (some features not available for FPGAs)

Schematic Editor

With the Schematic Editor, you can create a variety of schematic designs:

- all-schematic top-level designs
- top-level designs with instantiated HDL
- LogiBLOX, State Machine modules, and schematic modules that can be instantiated in HDL designs.

Major new features in the Editor include the following:

- enhanced bus taps connectivity
- improved Delete Symbol operation (Deleting a symbol leaves connected wires in place.)
- enhanced Design Rule Checking (DRC checks for connections of named signals to unnamed buses at the time these connections are made.)
- improved support for hanging wire terminators
- temporary symbol generation for missing symbols
- schematic tabs added to the main window to easily switch between pages. See the “Schematic Tabs” section in the *Foundation Series User Guide* for details.
- ARRAY symbol support for multiple instances of a symbol ($ARRAY parameter) when importing a Viewlogic schematic.
- improved behavior when dragging groups of bus taps
- Replace Symbol option for replacing all instances of a selected symbol
- Simulate Current Macro option. See the “Simulate Current Macro” section in the *Foundation Series User Guide.*
- improved annotation for renumbering symbol references when merging schematic sheets
- integrated LogiBLOX interface

State Editor

The State Editor allows you to do the following:

- graphically describe a state machine using the "bubble diagram" concept
- generate behavioral VHDL code from the diagram
- invoke the XVHDL compiler to convert the behavioral description into a gate-level EDIF netlist

The State Editor also supports the XABEL language. State actions may now include IF statements.

HDL Editor

In Foundation Series 1.4, you can initiate LogiBLOX within the HDL Editor (**`Synthesis`** → **`LogiBLOX`**).

If you received and installed Foundation Express, then Verilog is automatically enabled within the HDL Editor. When you use the Design Wizard to create a new project, the Verilog option displays in the Design Wizard - Language window. Like VHDL, Language Assistant templates and color coding are provided for Verilog. However, unlike VHDL, you cannot synthesize Verilog designs within the HDL editor. You must use Foundation Express to synthesize your Verilog designs. See the *Foundation Express Application Note Supplement* for information about Verilog design flows.

Simulator

The simulator can accept a hierarchical EDIF netlist as input, allowing a design's hierarchy to be manipulated, maintained, and viewed in the simulator. Some of the major new features of the simulator include the following:

- support for large designs with a maximum of 32000 modules in each hierarchical level
- Waveform Copy / Paste option for single waveforms
- dragging of signal transitions in the Waveform Editor. See the "Waveform Editing Functions" section of the *Foundation Series User Guide* for details.

- Simulate Macro Editor option. See the "Simulation Macro Editor" section of the *Foundation Series User Guide* for details.
- new windows for managing simulation formulas
- support for saving formulas and stimulators as ASCII Test Vectors
- improved display and notification of timing violations
- improved hierarchical simulation speed
- faster loading of large EDIF netlists

In addition to the improvements to the simulator, the new Simulation Macro Editor, which is based on the HDL Editor, lets you edit and run simulation scripts. To access the Simulation Macro Editor, from the Logic Simulator window, select **Tools** → **Script Editor**. To find out how to use the new editor, within the Macro Editor window, select **Help** → **SIM Macros Help**.

Library Manager

The XC3000 and XC5200 libraries are now available for Foundation customers. The XC7000 library has been removed.

Some features of the Library Manager user interface include the following:

- a toolbar with buttons for library management operations
- main window tabs for selecting libraries and components within those libraries
- color coding representing the type of library (system or user)
- resizable list columns
- list header area that allows you to sort the libraries and symbols based on any of the fields
- print preview function

Symbol Editor

The user interface of the Symbol Editor supports the following features:

- a toolbar that provides access to frequently-used Symbol Editor functions through a series of pushbuttons.
- Copy / paste functions which encourages easy creation of a new symbol by using an existing symbol as a template.
- improved pin type visibility control
- support for long names

Foundation Express (Optional)

Foundation Express allows Foundation customers to use Xilinx's new HDL synthesis technology (provided by Synopsys).

The major new features of Foundation Express include the following:

- high-quality synthesis for all Xilinx device families
  a) XC3000 and XC5200
  b) XC4000 up to XC40125
  c) XC4000XV
  d) Spartan
  e) XC9500
- VHDL and Verilog support
  a) improved Verilog black box support

     To instantiate a module in Verilog, you simply define an empty module containing the names and directions for the module's I/O pins and then instantiate the module. The instances of the module are automatically stored in the final netlist.

     You can also specify timing constraints on arbitrary timing paths, including multi-cycle paths. You can create a subpath by right-clicking on a path in the pre-optimized implementation.

b) updated faster HDL analyzers

The latest (V)HDL analyzer from Synopsys Design Compiler version 1997.08 contains bug fixes and additional construct support.

- graphical constraint entry

  a) customizable timing groups

  b) improved overall system performance requirements

- new Interactive Graphical Timing Analyzer

  Foundation Express includes a timing analyzer for timing report and debugging, which is built on the Paths and Ports constraint tables. After optimization, the timing verifier displays:

  - The timing for both path groups and subpaths (Paths page). Double click on the path icon to display or hide the subpaths of a timing path. Clicking a path or subpath shows the worst delays to all endpoints in the To group. Clicking an endpoint shows the complete critical path from the start to endpoints.
  - The input to clock and clock to output timing (Ports page) (Timing analyzer support is not available for the Xilinx XC9500.)

- automatic finite state machine (FSM) optimization

  Automatic FSM encoding is supported for enumerated types (VHDL); use the VHDL template to design your FSM, then choose One Hot or Binary encoding under **`Synthesis`** → **`Options`** → **`Project`**.

- improved graphical user interface

  The graphical user interface (GUI) has a new look and feel. In addition to the Design Sources window and the Chip window, the Error / Warnings window is always displayed. Several activities, such as entering constraints or viewing results, are started by clicking the right mouse button. Double clicking on icons expands or contracts the file or design hierarchy. For VHDL, other libraries than WORK can be created by clicking the right mouse button on the project icon, and selecting New Library.

- built-in editor source editor

  The editor interacts with analysis errors and lets you edit source files. You start the editor and open the file by selecting the design file icon, clicking the right mouse button, and selecting Edit File. You can use a menu of editing options by clicking the right mouse button.

- M1/XACT option

  You can target either M1 or XACT. Targeting M1 turns off the HBLKNM setting. HBLKNM is used on some look-up tables to improve the quality of result when using XACT. The switch to control this feature is under the Xilinx Options spreadsheet of the pre-optimized implementation.

- choice of optimization parameters

  You can synthesize a design for high speed or low area. The switch to control this feature is under the Create Implementation window. Also with the Low/High effort switch, you can control the CPU effort for the optimization engine. Low effort runs faster, though High effort provides better quality of results. The switch to control this feature is under the Create Implementation window.

- GSR Mapping for designs including black boxes

  Foundation Express can now infer the global set reset (GSR) even for design that contains black boxes (when allowed by the selected device). The switch, which is called Ignore Unlinked Cells During GSR Mapping, can be found by selecting **`Synthesis`** → **`Edit Constraints`** → **`Xilinx Options`** of the pre-optimized implementation.

- enhanced export report file

  The export report file contains much more detailed information on the pre-optimized as well as the post optimized design, such as: the design's hierarchy, the inferred operators, the cell count, the timing constraints, and the clock speed estimates.

If Foundation Express is installed, the HDL Editor provides color-coding and language assistance for the Verilog language. However, you cannot synthesize your designs within the HDL Editor. These designs must be synthesized using Foundation Express.

Help System

Complete online help and online documentation is provided with the Foundation Series software. The online help system can be invoked from the Project Manager's Help menu or from the Xilinx Foundation Series program group.

The online help for the design entry tools is provided in Windows help files on a per-application basis. The new online help system has an improved look-and-feel. In general, it is more consistent than the previous version and exhibits more extensive use of Windows 95 help features.

With this new version of the online help, Xilinx FPGA designers can find more information than they could previously, including a tutorial that teaches basic FPGA design techniques. In addition, the help system now includes links that take you directly to the Xilinx web site.

Online Documentation Suite

The online documentation for the design implementation tools is organized into books, and you use a different browser (not based on Winhelp), DynaText, to view it. Xilinx manuals are divided by subject matter, and several volumes can apply to one or more products.

Note: For more information on DynaText, see the "Troubleshooting" appendix in the *Foundation Series 1.4 Install and Release Document*.

Design Implementation Tools

The Xilinx implementation software offers many important features. These features are described in the following sections, first for FPGA devices, then for CPLD devices.

FPGA-specific Features

This release contains a suite of software that supports all available FPGA devices. Designs can easily be retargeted between various FPGA device families.

EDIF Netlists Supported

Third-party synthesis or schematic capture tools create various types of netlists. When using Foundation, the preferred netlist file format is EDIF; however, other netlist formats are supported. Refer to the following table for a list of the netlists supported by Foundation Series 1.4.

Table 1-1 Supported Netlist Formats

Netlists	Variations	
EDIF	Multiple variations are accepted, including:	
	SEDIF	EDN
	EDIF	
XNF	Multiple variations are accepted, including:	
	SXNF	XNF[a]
	XFF	XTF

a.XNF support has been retained to provide backwards compatibility with other tools, including previous releases of Xilinx XACT*step* software.

Netlist Optimization Built into Mapper

MAP, the mapping program, performs its own netlist optimization. Advantages to the customer include the following:

- faster, more reliable results
- no need to run a separate program
- improved optimization results

Improved MAP Reports

The new MAP reports have a new table of contents. Errors and warnings are grouped together into special sections at the beginning of the report. The report summary has also been moved to the beginning of the report. Three sections of the report are optional which reduces the length and file space requirements.

Place and Route Improvements

For Foundation Series 1.4, significant runtime improvements have been made to the PAR software. In addition, the Design Manager displays a single slider bar to define the PAR effort level. Select the Place and Route Tab from the Implementation Option window to see the new slider bar.

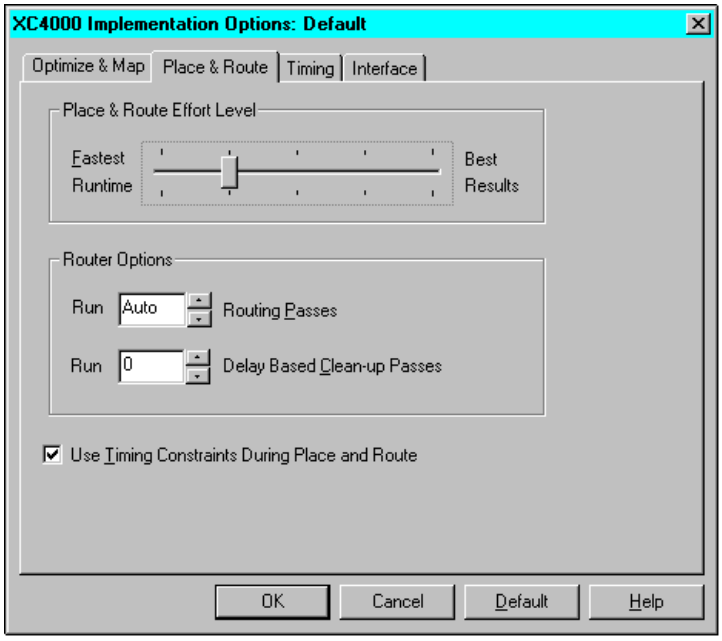


Figure 1-1 New Slider Bar

PAR reports contain a new Suggestions section with hints on how to improve design performance. PAR reports also provide improved resource utilization descriptions.

Static Timing Analysis After Each Major Process

You can analyze your design for non-resolved timing constraint, clock, and path problems by looking at its timing; use the interactive Timing Analyzer tool to perform this analysis.

You can perform analysis at three key places in the design process for FPGAs 1) after merging netlists and selecting a target architecture, 2) post-map, or 3) post-place-and-route. For CPLDs, you can perform static analysis after running your design through the CPLD fitter.

Post-MAP timing reports can be very useful in evaluating timing performance. Although routing delays are not accounted for, evaluating logic delays can provide you with valuable information about the design. On the other hand, post-PAR timing reports incorporate *all* known delays to provide a comprehensive timing summary. Foundation Series 1.4 static timing reports describe net fanout.

Multi-pass Place-and-Route

The place-and-route (PAR) software allows multiple place and route iterations to be run:

- on a single machine
- on various computers distributed on a wide-area network (WAN)
- on multiple machines in parallel

The multi-pass feature achieves optimum performance and efficiency, utilizing maximum CPU time to achieve design results more readily and quickly.

Re-entrant Routing

Once a place and route result is found that is close to meeting the desired specifications, notably those of timing, or is close to being completely routed, the implementation process can be re-entered to continue the routing process. This feature allows you to modify attributes and constraints during the routing stage of design implementation.

In addition to facilitating design changes, re-entrant routing significantly reduces CPU time of re-compiles.

EPIC Graphical Viewer/Editor

Editor for Programmable Integrated Circuits (EPIC), a graphical editor, provides a view of the physical implementation of your Xilinx design. The availability of EPIC means you no longer have to use XDE/EDITLCA. Important features of EPIC include the following:

- the ability to view CLB mapping, placement, and routing
- a timing analysis with critical paths highlighted
- the ability to modify the placement and routing of your Xilinx design
- the creation of custom macros to be incorporated into your Xilinx design

Flow Engine

The Flow Engine displays and then executes all the steps needed to implement a Xilinx design. The Flow Engine performs the following:

- translates the design netlist
- maps the logic to CLBs (FPGAs)
- places and routes the design (FPGAs)
- creates a configuration file which downloads a design to a part
- creates static timing reports and timing simulation netlists in the following formats: VHDL (Vital), Verilog, EDIF, or XNF

Timing Analyzer

The Timing Analyzer produces a report on the following:

- overall design performance
- timing specifications performance
- specific path performance

Timing analysis can be performed on the block delays of a just-mapped design or on the block and route delays of a design already placed and routed.

Hardware Debugger

The Hardware Debugger is useful for prototyping. The Hardware Debugger downloads a configuration file to a single FPGA or to a daisy chain of FPGAs through the XChecker, Serial, or Parallel download cables. When used with the XChecker cable, the Hardware Debugger can read back the state (logic levels) of the design signals inside the FPGA, and thus, enable in-circuit design debugging.

PROM File Formatter

The PROM File Formatter creates files for serial- or byte-wide configuration PROMs. Three formats are available: MCS-86, EXORmacs, and TEKHEX. The HEX format is also supported for microprocessor-based configuration. (The HEX file type is not considered a valid PROM file format because you cannot use it to program PROM devices.)

CPLD-specific Features

The CPLD fitter supports all available XC9500 devices. For information on how to create and process XC9500 designs using this software, refer to the online help provided for the CPLD fitter.

CPLD Implementation Software

By default, the CPLD fitter automatically takes advantage of special CPLD architecture features, including global clocks, global output enables, global set/reset, D-type and T-type flip-flops, and XOR gates.

The fitter also performs general timing optimization to reduce overall path delays in your design. If desired, you can disable any of these features.

The CPLD fitter produces a comprehensive report listing the resources used, the resources remaining, the resulting device pinout, and the allocation of macrocell resources for your design. The fitter also produces a static timing report showing the calculated worst-case timing for the logic paths in your design.

The fitter has an advanced pin-locking feature for XC9500 designs. The software automatically selects initial pin locations for your I/O signals that give you the greatest flexibility to iterate your design after committing your pinout. Each time the fitter successfully implements your design, it stores your pinout in a "guide file" that will optionally lock your pinout during subsequent iterations.

Comprehensive Online Help

The Foundation Series online help system contains the information you will need to design XC9500 CPLDs. In addition to providing links to the information available about all the design entry and implementation tools, the Foundation online help provides in-depth information on the following CPLD topics:

- CPLD design techniques and application notes
- tutorial teaching CPLD design flow
- XC9500 JTAG programming
- reference information on schematic library, attributes and implementation options

JTAG Programmer Interface Software

This release supports in-system downloading of programming files to supported XC9500 devices via the JTAG test access port. The interactive JTAG Interface software also provides for testing of on-chip logic and is compatible to the IEEE 1149.1 standard. The software also includes BSDL files for use with third-party boundary scan and ATE systems.

Design Constraints/Properties

Foundation Series 1.4 software provides a revised set of design properties (also attributes and/or constraints) that is consistent across design entry methods and between device families (FPGA and CPLD). Several new properties have been added to control CPLD design implementation, including global clock/OE/set/reset allocation, logic collapsing, and improved timing specifications.

Properties can be specified either in the source design or via the new user constraint file (UCF), which replaces the CST file used in previous releases. Refer to the "Using Constraint Files" section of the "Foundation Overview" chapter for more information about user constraint files. Alternatively, refer to the Foundation online help for descriptions of the new property names used in this release.

Chapter 2

Setting Up the Foundation Tools

This chapter lists the system requirements for the Foundation Series 1.4 Xilinx design tools software. This chapter discusses the recommended machine types and memory requirements to comfortably run the Foundation Series 1.4 software. Also included are general instructions for installing the software, contacting customer support, and obtaining and installing the necessary authorization codes and licenses.

This chapter contains the following sections:

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

Installation Notes

Ensure the optimum use and operation of your new design tools by installing Foundation Series 1.4 on the recommended hardware with sufficient memory (RAM and swap). If you experience problems with either the installation, operation, or verification of your installation, please contact the Xilinx Technical Support hotline. Refer to the "Technical Support" section of this chapter for specific phone numbers and email addresses to use.

Supported Platforms and Machine Requirements

The Foundation Series 1.4 software is a PC-only release. Foundation runs on either Windows 95 or Windows NT (version 4.0 or later). The following list shows the type of PC you should have to perform designs for Xilinx FPGAs or CPLDs.

- Pentium Pro Processor®
- Windows 95® or Windows NT 4.0®
- 120 MHz clock speed
- Memory—32 MB to 64 MB (dependent on device)
- Swap Space—48 MB to128 MB (dependent on device)
- 500 MB disk space (for 1 device family only)
- 2 GB hard disk (for all device families)
- SVGA 17" monitor
- 4x CD-ROM drive
- Keyboard
- Mouse

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

Swap file size requirements also vary with the design and constraint set size. By default, Windows 95 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 machines, or machines with less than the recommended RAM and/or swap space, will 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 2-1 Memory Requirements

Xilinx Packages	RAM	Virtual Memory (Swap Space)
Base Base with VHDL	48 MB	64 MB
Standard Standard with VHDL	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 “Installation” section of the *Foundation Series 1.4 Install and Release Document* for complete details on installation.

1. Insert the Design Entry Tools CD. If your system has the Auto Run feature enabled, the Foundation Master Installer will start automatically. Select “Install Design Entry Tools” to begin installation.

   If you do not have Auto Run enabled on your system, you can bypass the Master Installer utility by running the Setup program (found in the “setup” directory on the Design Entry Tools CD-ROM) directly.

2. Follow the instructions on the screen to install the design entry tools; if you wish, you can wait until installing all portions of Foundation before restarting.
3. If you are installing Foundation from the Master Installer (described in Step 1 above), you can click on "Install Design Implementation Tools" to begin installation of this set of tools and the related DynaText documentation files. If you make this selection, you will be prompted to insert the Design Implementation Tools CD. After inserting the CD and clicking on OK, the installation will start.

   Alternatively, you can initiate the Implementation Tools setup program yourself by inserting the Implementation Tools CD-ROM and running the setup program located in the root directory of the CD.
4. 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 InstallShield program will inform you if you need to reboot.
5. (Optional) Insert the CD-ROM labeled "Esperan® MasterClass Lite VHDL Tutorial." Run "setup.exe" located in the CD-ROM root directory. (This CD is included only with FND-BSX and FND-EXP packages.)
6. (Optional) Insert the CD-ROM labeled "Foundation Express". Run "setup.exe" located in the CD-ROM root directory. Install this CD if you wish to use the Foundation Express software for synthesizing VHDL and Verilog designs files.

Network Compatibility

The Xilinx installation program supports only TCP-IP style networks. Novell is *not* a TCP-IP style network.

Note: If you are using XABEL, the entire Foundation software package must be installed on your local PC's hard drive. The XABEL compiler does not run reliably over a network.

Obtaining and Setting Up Licenses

Before running your Foundation software, you will need to obtain a license from Xilinx. To obtain a 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 in a license.dat file.

You can also obtain a license by accessing the online Web tool or by contacting your local Xilinx Customer Service representative. If you request your license via fax, please fill out the form that is located behind the front cover of this *Quick Start Guide*. For details about licensing, consult the "Setting Up Security" chapter in the *Foundation Series 1.4 Install and Release Document*.

Customer Service

Use the following information for contacting your local Xilinx Customer Service representative.

United States and Canada

Monday-Friday, 8:00am to 5:00pm Pacific Standard Time
voice 800-624-4782, and facsimile 408-559-0115.

Europe

Monday-Friday, 9:00am to 5:30 pm United Kingdom time—English speaking only.

Country	Telephone	Facsimile
United Kingdom	01932-333550	01932-828521
Belgium	0800-73738	
France	0800-918333	
Germany	0130-816027	
Italy	1677-90403	
Netherlands	0800-918333	
Other European Locations	(44) 1932-333550	(44) 1932-828521

Other International Countries

Country	Telephone	Facsimile
Japan	81-3-3297-9153	81-3-3297-9189
Southeast Asia / ROW	Contact local distributor	Contact local distributor

If you reside in an international country not listed, please contact your local distributor.

Technical Support

The following are Xilinx hotline access numbers.

Location	Telephone	Electronic Mail
U.S. and Canada	1-800-255-7778	hotline@xilinx.com
Japan	81-33-297-9163	jhotline@xilinx.com
France	33-1-3463-0100	frhelp@xlinx.com
Germany	49-89-9915-4930	dlhelp@xilinx.com
United Kingdom	44-1932-820821	ukhelp@xilinx.com
To contact factory	1-408-879-5199	hotline@xilinx.com

Xilinx sells a Student Edition of the Foundation Series software. If you are using this version of the software, you are requested to not call for technical support. Instead, visit the following web site and follow the instructions there to get answers to your questions and obtain technical support.

http: / / www.xilinx.com / programs / univ.htm

Chapter 3

Foundation Overview

This overview explains the basic concepts and design flow of the Foundation Series 1.4 release as it spans the flow from netlist to final PROM file. This chapter contains the following sections:

- "Design Flow" section
- "Using the Foundation Design Entry Tools" section
- "Using Foundation Express (Optional)" section
- "Using the Foundation Design Implementation Tools" section
- "Using Constraint Files" section
- "Guiding Implementation" section
- "Static Timing Analysis" section
- "Creating Simulation Files" section
- "Downloading a Design" section
- "Advanced Implementation Flows (FPGAs Only)" section

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.

Design Flow

The Foundation Series design tools interface supports the following design flows:

- top-level schematic entry with the Xilinx Unified Libraries components, LogiBLOX symbols, or both
- schematic entry with Unified Library components with some components expressed as HDL or state machine macros
- all-VHDL design
- mixed-VHDL designs with schematic-based, state machine, or LogiBLOX instantiated components
- finite state machine diagram entry

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

Note: If you purchased Foundation Express, then this HDL software tool supports designs having top-level HDL descriptions (either VHDL and Verilog). Refer to the *Foundation Express Application Note Supplement* for details.

The following two figures illustrate the basic design flow for FPGAs and CPLDs. For detailed design flow illustrations, refer to the "Design Flow" section of the *Foundation Series User Guide*. If you purchased Foundation Express, consult the *Foundation Express Application Note Supplement* for design flow information.

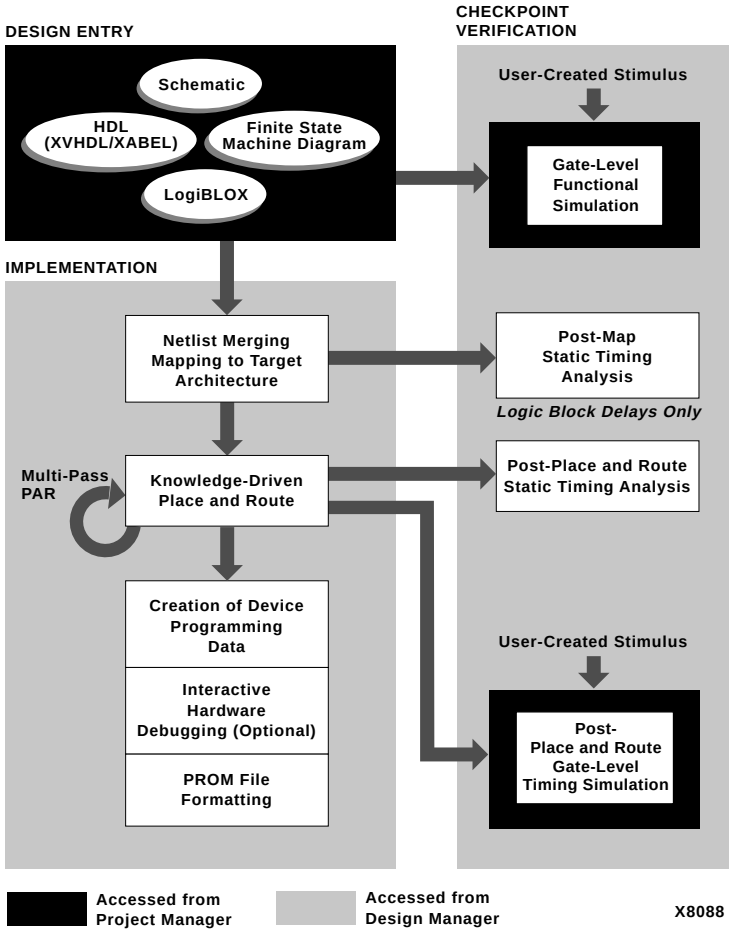


Figure 3-1 Foundation Overall Design Flow for FPGAs

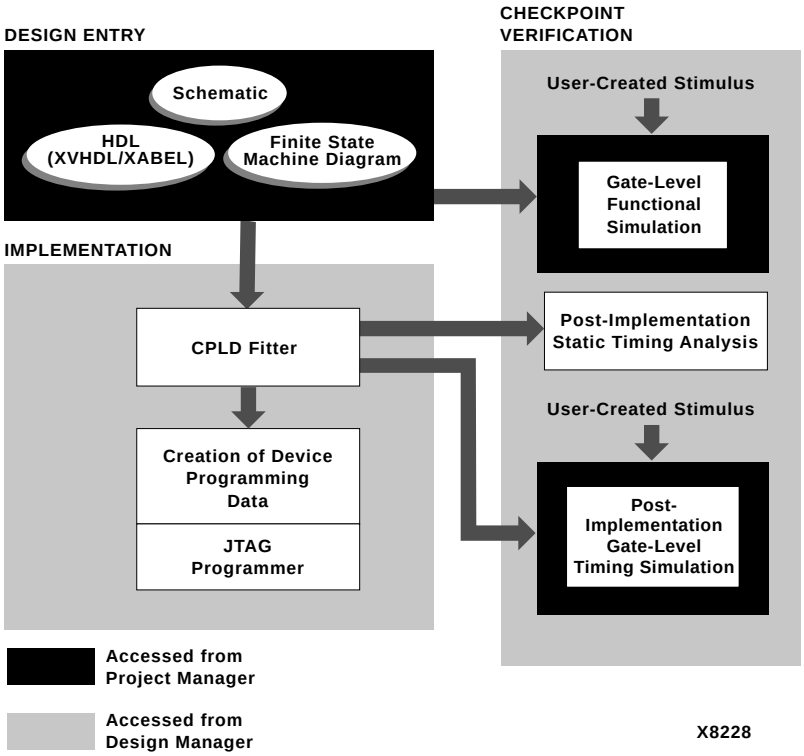


Figure 3-2 Foundation Overall Design Flow for CPLDs

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 on the Project Manager icon in the Foundation Series program group. The icon to click is shown in the following figure.

The Foundation design tools environment (the Project Manager) is shown in the following figure.

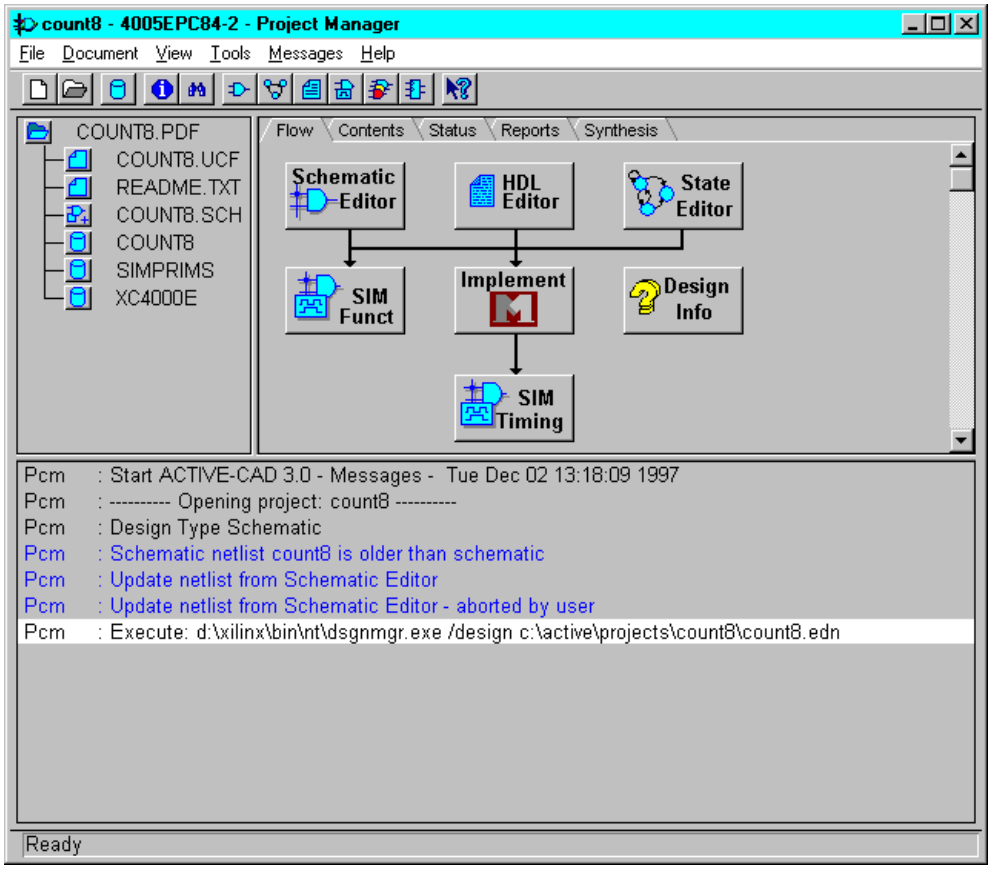


Figure 3-3 Foundation Project Manager

If this is the first time you have opened the Project Manager, a dialog box will pop up, asking if you want to open an existing project or create a new one.

To create a new project, follow these steps:

1. Open the Foundation Project Manager.
2. Select **File** → **New Project**.
3. Enter a name for the project. Change the directory for the project, if desired, by using the Browse button.
4. Choose the appropriate family, part, and speed grade.
5. Click OK.

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 in the *Foundation Series User Guide*.

Creating Top-Level Schematic Designs

You can create a variety of top-level designs:

- all schematic designs
- schematic designs with instantiated HDL macros
- schematic designs with instantiated LogiBLOX modules
- schematic designs with instantiated state machine macros

For a detailed description of the procedures for creating these types of designs, refer to the "Top-level Schematic Designs" section in the *Foundation Series User Guide*.

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

Creating Top-level VHDL Designs

You can create a variety of top-level schematic VHDL designs:

- all-VHDL designs
- designs with instantiated schematic macros
- designs with instantiated state machine macros
- designs with instantiated Xilinx Unified Library components
- designs with instantiated LogiBLOX modules
- designs with instantiated netlist modules

For a detailed description of the procedures for creating these types of designs, refer to the "Top-level VHDL Designs" section in the *Foundation Series User Guide.*

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

If you purchased Foundation Express, you can also use this software to create your VHDL designs. For information on how to create top-level VHDL designs w with Express, refer to the *Foundation Express Application Note Supplement*.

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
- a module in a schematic
- a module in VHDL

For a detailed discussion of the design steps, refer to the "State Machine Designs" section in the *Foundation Series User Guide.*

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

Using Foundation Express (Optional)

You can create VHDL and Verilog designs if you purchased the optional Foundation Express package. For details about this separate package, refer to the *Foundation Express User Guide* and the *Foundation Express Application Note Supplement.*

Using the Foundation Design Implementation Tools

This section describes the design implementation tools.

Benefits of Using the Design Manager

The Xilinx Design Manager is the graphical interface that manages the implementation versions and revisions created during the design process. Results of these implementations are made available in reports and may be accessed through the Design Manager's Report Browser.

The Design Manager also provides on-screen pushbutton access to the other Xilinx tools, such as the Flow Engine, Timing Analyzer, PROM File Formatter, EPIC Report Browser, and various navigation and information tools.

Starting the Xilinx Design Manager

To start the Xilinx design implementation tools, click on the Implement M1 button in the Project Flowchart area, shown below.

The Foundation Design Manager is shown in the following figure.

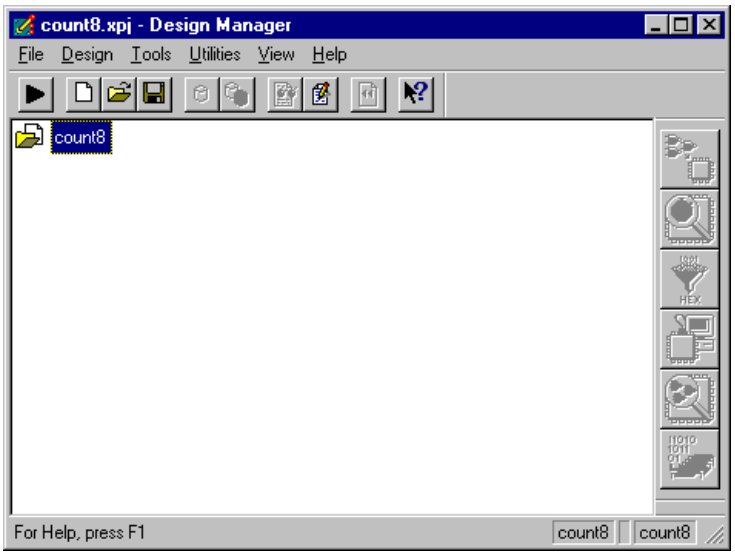


Figure 3-4 Foundation Design Manager

Implementing a Design

From the Design Manager menu (refer to the following figure), select **`Design`** → **`Implement`**.

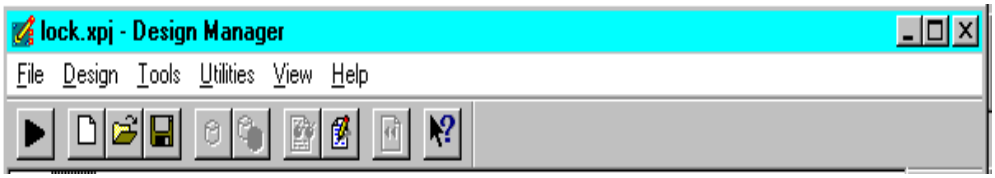


Figure 3-5 Design Manager Menus and Toolbar

In the Implement dialog, select the part and click on Run. The Design Manager automatically creates a new version and revision. Additional versions are created when the netlist is modified and re-implemented. Also, additional revisions are created when the same netlist is re-implemented with new options or constraints.

The Design Manager invokes the Flow Engine to process the design. The Flow Engine interface prominently displays the status of each phase of the design, as shown in the following figures.

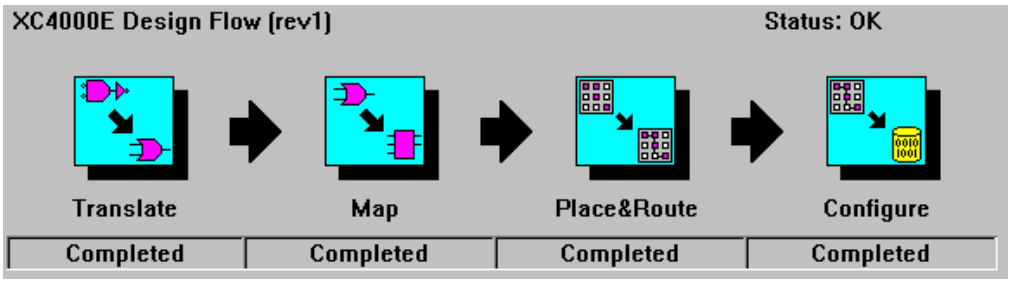


Figure 3-6 Flow Engine Shows All Design Segments Completed (FPGAs)

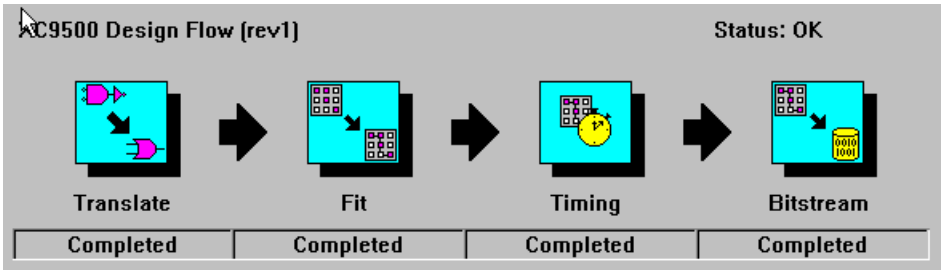


Figure 3-7 Flow Engine Shows All Design Segments Completed (CPLDs)

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 Map. Map optimizes the gates and trims unused logic in the merged NGD netlist. Map 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. The mapping process is accomplished by running the MAP executable. For more information about MAP, refer to the "MAP—The Technology Mapper" chapter in the *Development System Reference Guide*.

Place and Route (FPGAs)

Once 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. The place and route (PAR) process is accomplished by running the PAR executable. For more information about PAR, refer to the "PAR—Place and Route" chapter in the *Development System Reference Guide*.

CPLD Fitter (CPLDs)

The CPLD fitter implements designs for the XC9500 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 for XC9000), and timing simulation database (*design_name*.nga).

For detailed information about implementing CPLD designs, refer to the online DynaText books, *CPLD Schematic Design Guide* and *CPLD Synthesis Design Guide*. Also refer to the Foundation online help.

Configure

After place and route, the Flow Engine translates the physical implementation into a binary stream. The binary stream is used to program the FPGA. The binary stream is saved as a configuration file (.BIT) using the BitGen executable. For more information about the BitGen executable, refer to the "BitGen" chapter in the *Development System Reference Guide*.

Interpreting the Reports

The reports provide information on logic trimming, logic optimization, timing constraint performance, and I/O pin assignment. To access the reports, go to the Report Browser. (To open the Report Browser, select the Design Manager command, **`Utilities`** → **`Report Browser`**). To open a particular report, double click on its icon, as shown in the "Report Browser" figure, below.

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

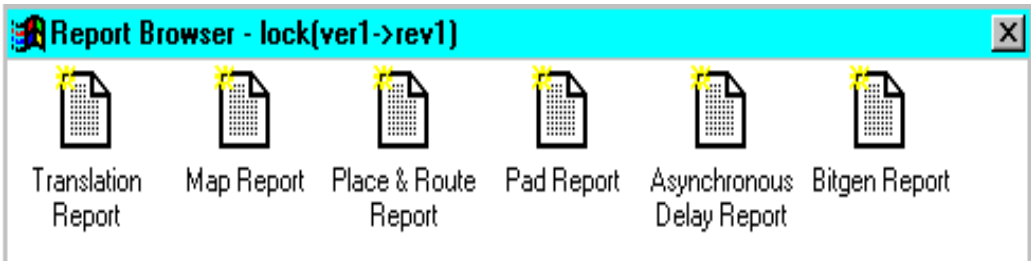


Figure 3-8 Report Browser

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:

- Erroneously 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. The placer score measures the "goodness" of the placement. Lower is better. The score is strongly dependent on the nature of the design and the physical part that is being targeted, so meaningful score comparisons can only be made between iterations of the same design targeted for the same part.
- The Number of Signals Not Completely Routed should be zero for a completely implemented design. If non-zero, you may be able to improve results by using re-entrant routing or the multi-pass place and route flow.
- The timing summary at the end of the report details the design's asynchronous delays. For information on timing constraint performance and synchronous delays, refer to the "Static Timing Analysis" section later in this chapter.

Pad Report

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. This section of the Pad Report can be cut and pasted into the .PCF file after the SCHEMATIC END; statement to preserve the pinout for future design iterations.

Selecting Options

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

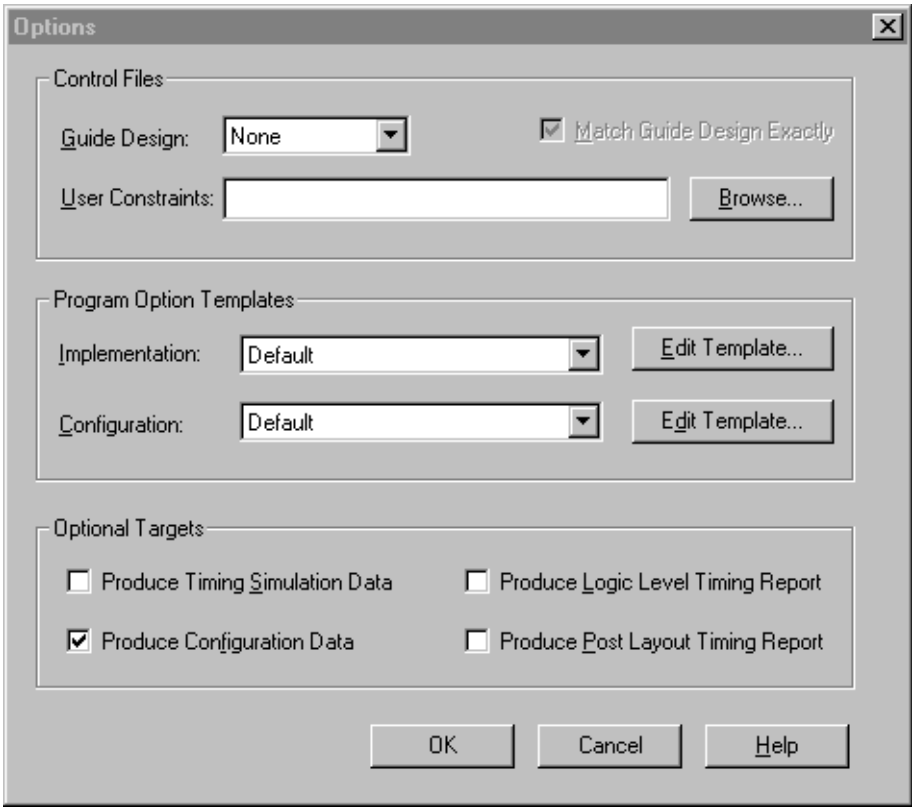


Figure 3-9 Flow Engine Options Dialog Box

You can have multiple templates in a project. By choosing a template, you are choosing an implementation or configuration style. To access the options and templates

- Select the Options button in the Implement dialog or from the Flow Engine menu select **`Setup`** → **`Options`**.
- In the Program Option Templates portion of the Options Dialog, select the Edit Template button for Implementation or Configuration to access the associated template.
- From the Design Manager menu, select **`Utilities`** → **`Template Manager`**.

The default options settings are able to accommodate most implementations. For information on the options, select **Help** → **Contents** from the Design Manager menu.

Using Constraint Files

The design implementation tools allow you to 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 to a synthesis tool, or listing them in a user constraint file. 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 create a user constraints file *design_name*.ucf.

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 Options dialog.

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

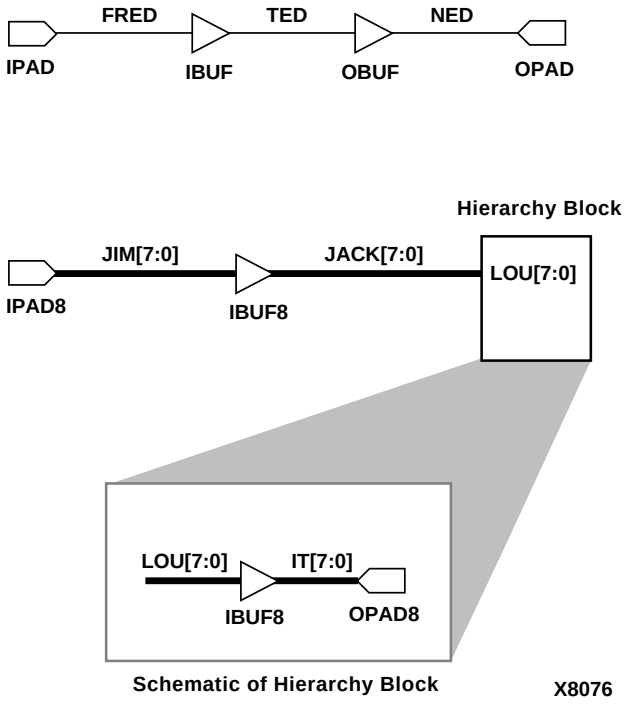


Figure 3-10 Locking I/Os to Pin Locations

```
# This is a UCF comment

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

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

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


# Lock the input pins

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

# Lock the output pins

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

For more information on constraint precedence, refer to the "Constraint Precedence" section in the *Foundation Series User Guide.*

This example shows how to specify timing constraints.

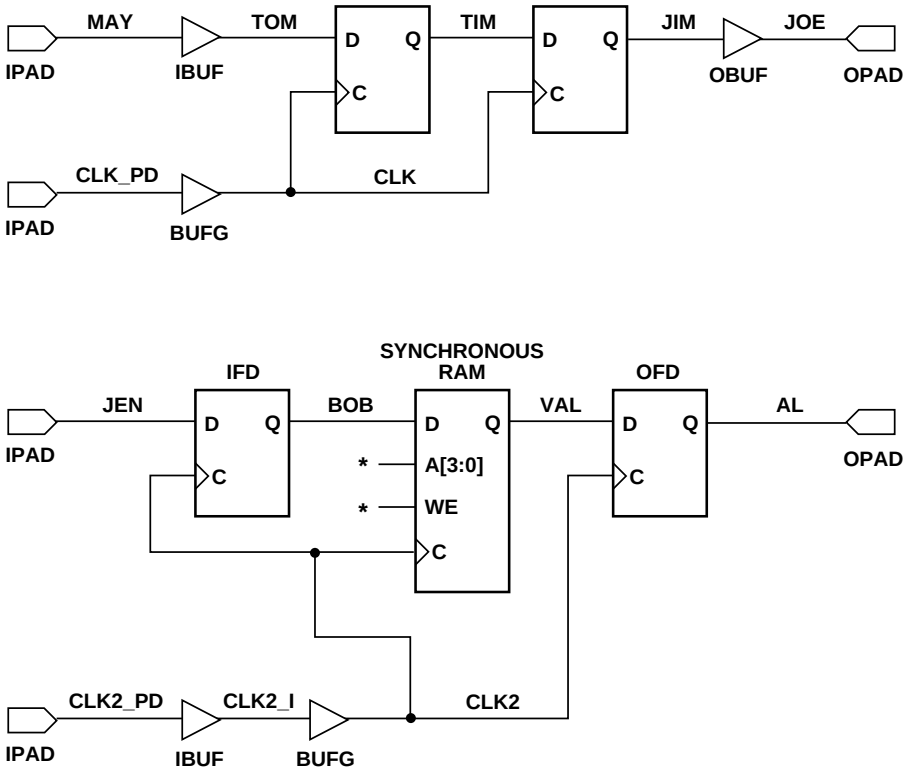


Figure 3-11 Specifying Timing Constraints

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

# This is a comment

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

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

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

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

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

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

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

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

TIMESPEC TS02=FROM : CLK2_LOADS : TO : CLK2_LOADS=30Mhz;

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

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

Guiding Implementation

In the Foundation Series 1.4 tools, guiding is used during map, place, and route. Guiding a design can significantly reduce runtimes, since less processing has to occur.

During a typical design process, a design is modified and implemented many times. The changes are such that from one implementation to the next there are many parts of the design that do not change. Guiding a design accelerates iterative implementations by reusing the unchanged sections from a previous implementation on current implementations. The software therefore only has to spend time generating implementations for sections of the designs that have changed.

To select a previous implementation to guide a current implementation, open the Options dialog. In the Guide Design dropdown list, you can select previously implemented revisions, Project Clipboard, Custom, or None.

The Project Clipboard is used to save the guide data of revisions that are being overwritten. Guide data can be saved to the Project Clipboard by selecting the Copy *previous_revision* Guide Data to Project Clipboard button in the Implement dialog. NCD files created outside of the project can be used for guiding by selecting the Custom option. In the Custom dialog popup, be sure to enter 1) a mapped NCD file, and 2) a placed and routed NCD file. If guide files aren't needed, select None.

Exact Guide Mode

When guiding in exact mode, the unchanged logic is not modified in any way. This mode is fastest, but least flexible. Use this mode if the design iteration requires only minor changes. Exact mode is the default mode. It can be selected by having the Match Guide Design Exactly option enabled in the Options dialog.

Leveraged Guide Mode

When guiding in leveraged mode, the mapping, place, or route of the unchanged logic can be modified if the tools need to make layout changes to accommodate new logic. Use this mode if significant changes have occurred, such as re-synthesis of an entire hierarchical block.

Leveraged mode is automatically selected when the Match Guide Design Exactly button is *not* selected in the Options dialog.

Static Timing Analysis

Timing analysis can be performed at several stages in the implementation flow to estimate delays. A post-map timing report can be generated to evaluate the effects of logic delays on timing constraints, clock frequencies, and path delays. A post-place-and-route timing report that incorporates both block and routing delays can be generated as a final summary of the design's timing constraints, clock frequencies, and path delays. Detailed timing constraint, clock, and path analysis for post-map or post-place-and-route implementations are done with the interactive Timing Analyzer tool.

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 once routing delays are added. In fact, if the logic-only-delays exceed the total allowable delay for a path or constraint, then the place-and-route process can be skipped altogether, since the routing delays will only cause the path's timing to degrade.

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 (<15%) than the total allowable delay for a path or timing constraint, then very low placement effort levels can be used by the place and route software. 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.

Edit the implementation template to modify the placer effort level. For information on re-entrant routing or multi-pass place and route, see the "Advanced Implementation Flows (FPGAs Only)" 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. For a post-map report, select the Produce Logic Level Timing Report button. For a post-PAR report select the Produce Post Layout Timing Report button.
2. To modify the reports to highlight path delays or paths that have failed timing constraints, 1) edit the template implementation, 2) select the timing tab, and 3) select a report format.
3. Once MAP or PAR has completed, the respective timing reports appear in the Report Browser.

Detailed Timing Analysis

To perform detailed timing analysis, select **`Tools`** → **`Timing Analyzer`** from the Design 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:

- Choose sources. From the Timing Analyzer menu, select **Path Filters** → **Path Analysis Filters** → **Select Sources**.
- Choose destinations. From the Timing Analyzer menu, select **Path Filters** → **Path Analysis Filters** → **Select Destinations**.
- To create a report, select **Analyze** → **All Paths**.

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

Once the design is implemented, a timing simulation can be performed 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?

The design implementation tools allow you to create simulation data after each major processing step. This means that you can create functional simulation netlists, after the design has been merged together by NGDBuild in the Translate process, and timing simulation netlists after the design has been placed and routed by PAR for FPGAs or fitted by the CPLD fitter for CPLDs. Additionally, for FPGAs, you can create simulation data after the design has been mapped or after the design has been placed but not routed. 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 near the start of this chapter.

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

Post-map simulation allows you to 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. Post-place simulation can be used 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** → **Checkpoint Simulation** from the Foundation Project Manager, and choose the appropriate netlist to be simulated.

Creating Functional Simulation Data

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

For designs that include underlying netlists (XNF or EDIF), the design must first be "translated" in the Xilinx Design Manager 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. Click on the Xilinx M1 button in the Foundation Project Manager to invoke the Design Manager.
2. Select **Design** → **New Version** and then **Design** → **New Revision**.
3. With the new revision highlighted, go into the Flow Engine by clicking on the Flow Engine toolbar icon.
4. From within the Flow Engine, select **Setup** → **Stop After** and then choose the Stop After Translate option.

5. Click OK, then click the Run button in the Flow Engine.
6. After Translate is complete, go back to Foundation Project Manager and select **`Tools`** → **`Checkpoint Simulation`**.

   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 User Guide*.

For additional information about functional simulation, see the "Performing Functional Simulation" section of the *Foundation Series User Guide*:

Creating Timing Simulation Data

Before performing timing simulation, ensure that you have generated a timing annotated netlist and created timing vectors. See the "Post-Implementation Timing Simulation" chapter in the *Foundation Series User Guid*e for details.

To create timing simulation data, open the Xilinx Design Manager Implementation Options dialog, and select the Produce Timing Simulation Data option. Click OK. After the Implementation process is complete, return to the Foundation Project Manager, and click on the **`SIM Timing`** button, shown below. This invokes the Simulator and loads the timing simulation netlist.

For additional information about simulation, refer to the "Performing Functional Simulation" section of the *Foundation Series User Guide*.

Downloading a Design

An implemented FPGA design can be downloaded directly from your PC using the Hardware Debugger program and the XChecker cable.

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 either the Hardware Debugger or the XChecker cable, see the *Hardware Debugger Reference/User Guide*.

An implemented CPLD design can be downloaded from your PC using the JTAG Programmer. See the *JTAG Programmer Guide* for details.

Creating a PROM

An FPGA or daisychain 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 daisychain 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 and the XChecker 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, the Hardware Debugger, or the XChecker cable, see the *Hardware Debugger Reference/User Guide*.

Advanced Implementation Flows (FPGAs Only)

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. PAR options can be varied in many different ways; the following sections discuss the most common strategies.

Re-entrant Routing

PAR can take an implemented design as an input and re-route it. If your design is placed but not routed, PAR will use the placement and just spend time routing the design. If your design is partially routed, PAR will use the existing placement and routing and only spend time routing the unrouted signals. 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 come up with 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,

1. In the Design Manager, select the implemented revision and then select the Flow Engine button in the toolbox.
2. In the Flow Engine, select the **`Setup`** → **`Advanced`** menu.
3. In the Advanced dialog, select the Allow Re-Entrant Route button, 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 be performed. 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. No predictable criteria can be suggested to choose one style over the other. The best methodology is to select no more than three passes for each (in most cases, a single pass for each is sufficient), and use the PAR report to determine which was most effective and try using more clean-up passes of that style.

4. Click on OK (in the Advanced dialog) 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.
5. If you are specifying timing or location constraints, you may want 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. Since your design is already implemented, step back to the beginning of Place and Route using the Step Backward button at the bottom of the Flow Engine, and then click the button to start again.

Multi-pass Place-and-Route

If a design has not completed routing or has not yet met timing constraints, then you can use PAR to perform a more extensive search for a solution. PAR can produce multiple placed and routed revisions, each revision with varying implementations. PAR scores each implementation, choosing the best revisions based on the score. By choosing the best implementation from a large population, PAR is more likely to find a solution that meets your requirements.

If you are using the Foundation Series 1.4 software on a network, then to significantly reduce runtime, the place and route passes can be run in parallel by executing each pass on a separate machine. To initiate Multi-Pass Place-and-Route:

1. In the Design Manager, select a version (not a revision), and then from the menu choose **Design** → **FPGA Multi-Pass Place & Route**.
2. In the FPGA Multi-Pass Place and Route dialog, select a value for the Starting Strategy. The Starting Strategy is a value that initializes the place-and-route algorithms. Each iteration receives an incremented value of the starting strategy. For initial runs, set the Starting Strategy to 1, since 0 was used in your previous single-pass run.
3. Select the number of iterations to run.
4. Select the number of iterations to save. Based on the design score, only the files from the best runs are saved.
5. Click on OK to launch the multi-pass place and route process.

Note: If you are running on an NT workstation and want to run on multiple NT workstations, select a nodelist file. A nodelist file is a user-created ASCII file that lists the names of the workstations on which you want to run. Each name should be on a separate line. Do not include tabs or spaces in the file.

Chapter 4

In-depth Tutorial

This chapter guides you through a typical FPGA design procedure using a design called Calc, a 4-bit processor with a stack. The design example used in this tutorial demonstrates many system features that you can apply to your own designs.

In the first part of the tutorial, you use the Foundation design entry tools to complete the design. Next, you use the Logic Simulator to perform functional simulation. In the third part, you use the Xilinx Design Manager to implement the design. Next, you verify timing with the Logic Simulator and download the bitstream to a Xilinx demonstration board. Finally, an incremental design change is made and the design is recompiled to reflect the change.

Note: Although this tutorial describes the steps for creating and processing an FPGA design, most of the steps can be applied in the same manner to CPLD designs.

This chapter includes the following sections:

- "Design Flow" section
- "Getting Started" section
- "Targeting an XC9500 Device" section
- "Completing the Calc Design" section
- "Controlling Implementation from the Schematic" section
- "Modifying the Design for Non-XC4000 Family Devices" section
- "Using LogiBLOX (Optional)" section
- "Using the State Editor (Optional)" section
- "Using the HDL Editor and XVHDL (Optional)" section
- "Other Special Components (Optional)" section

- "Using a Constraints File" section
- "Functional Simulation" section
- "Using the Design Implementation Tools" section
- "Timing Simulation" section
- "Examining Routed Designs with EPIC" section
- "Verifying the Design Using a Demonstration Board" section
- "Making Incremental Design Changes" section
- "Further Reading" section

Design Flow

An incremental design methodology is described in this tutorial. In incremental design, the design is processed, a small change is made, and the design is processed again. Place-and-route information from the previous design cycle is used to constrain subsequent cycles. When this method is used, design performance remains relatively stable across design versions. Also, place-and-route time is considerably reduced since information from previous design cycles is leveraged.

The tutorial design can be targeted for an XC4000E or XC9500 device. You can use a Xilinx demonstration board to test the functionality of the design. Make sure your demonstration board and software support your selected device. To determine compatibility, refer to the release documents that came with your software package.

The two figures immediately following show the flow of the design once you are past the design entry phase and are either fine-tuning or determining the feasibility of your design. This phase of the design process is called the design implementation phase. The figures show, respectively, 1) FPGA (design implementation step) and 2) CPLD (design implementation step).

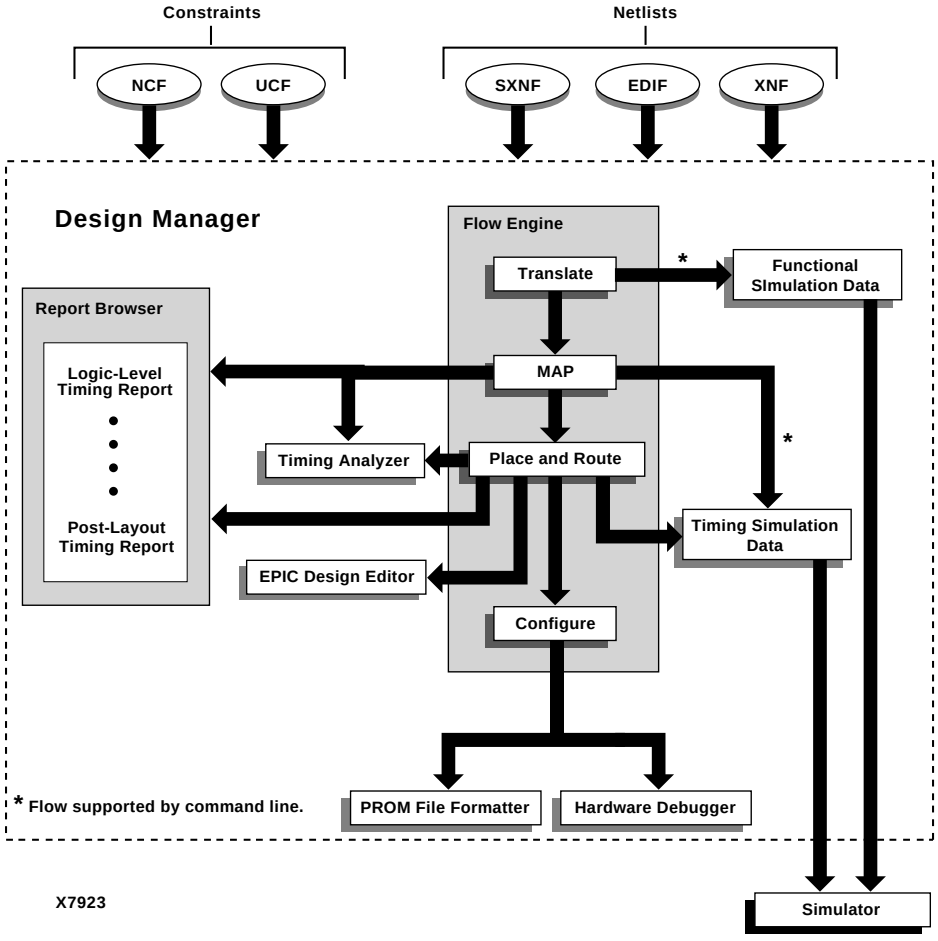


Figure 4-1 Foundation Series 1.4 Design Flow (FPGA Design Implementation Step)

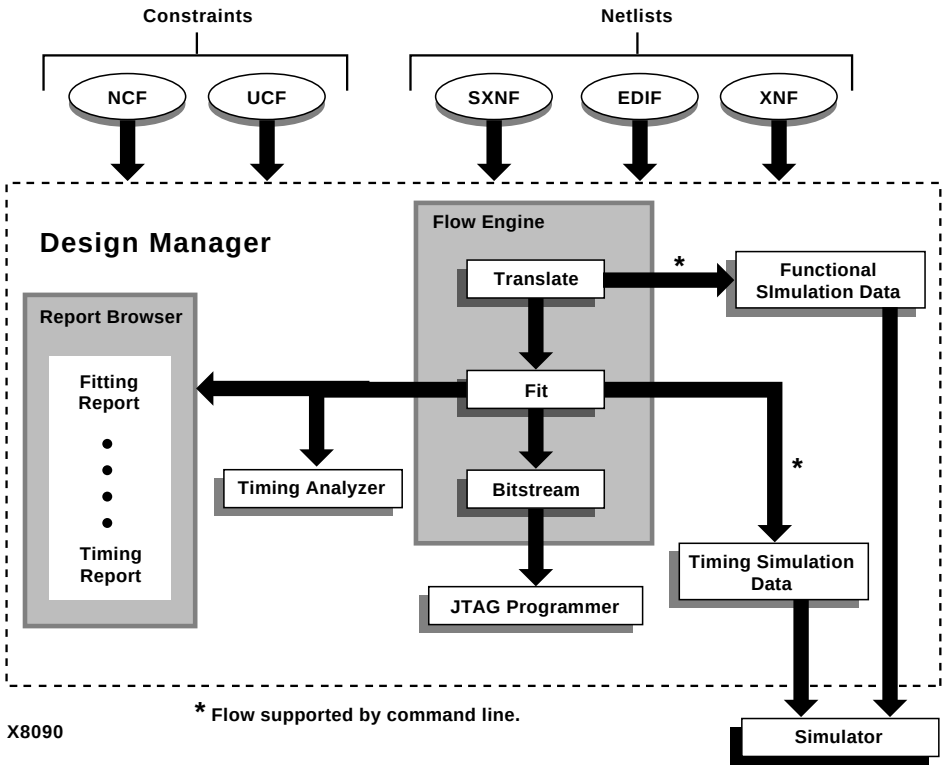


Figure 4-2 Foundation Series 1.4 Design Flow (CPLD Design Implementation Step)

Note: See the "Design Flow" section of the "Foundation Overview" chapter for more information about Xilinx Foundation design flows.

Getting Started

The following subsections describe the basic requirements for running the tutorial.

Nomenclature

In this tutorial, the following terms are used:

- "XC4000 family" includes XC4000E/L, XC4000EX, XC4000L, XC4000XV, and XC4000XL devices.
- "Right-click" means click the right mouse button. Unless specified, all other mouse operations are performed with the left mouse button.

Throughout this tutorial, file names, project names, and directory names (paths) are specified in lower case, and the design is referred to as Calc.

Required Software

The Xilinx Foundation Series package, Version 1.4 or later, is required to perform this tutorial.

Note: Some optional sections of the tutorial require specific Foundation Series features, available only with certain licensed packages.

Installing the Tutorial

This tutorial assumes that the software is installed in the default location c:\fndtn\active. If you have installed the software in a different location, substitute your installation path for c:\fndtn\active.

The tutorial projects are optionally installed (as sample projects) when you install the Foundation Series software. If you have installed the software, but are not sure whether the tutorial projects were installed, check for directories named c:\fndtn\active\projects\calc*. These directories contain the various tutorial files.

Note: Refer to the "Setting Up the Foundation Tools" chapter for installation instructions. For even more detailed instructions, refer to the *Foundation Series 1.4 Install and Release Document*.

Tutorial Project Directories and Files

During the tutorial installation, the c:\fndtn\active\projects\calc project directory is created, and the tutorial files are copied into this directory. This directory is missing some files because you will create them in the tutorial. However, solutions projects with all input and output files are also provided; they are listed in the following table.

Table 4-1 Tutorial Project Directories

Directory	Description
calc	Main tutorial project directory
calc_4ke	Solution for XC4003E-PC84
calc_9k	Solution for XC95108-PC84
calc_blx	Solution for XC4003E-PC84 using LogiBLOX, ABEL, and VHDL

The three solutions project directories contain the design files for the completed tutorial, including schematics and the bitstream file.To conserve disk space, some intermediate files are not provided. Do not overwrite any files in the solutions directories.

The calc project contains an incomplete copy of the tutorial design. A few intermediate files are also included, and you will create the remaining files when you perform the tutorial. As described in a later step, you can copy the Calc project to another area and perform the tutorial in this new area. The following table lists some of the directories and files in the calc_4ke solution project directory.

Table 4-2 Directories/Files in the Calc_4k Project Directory

Directory or File Name	Description
calc.sch	Top-level schematic
calc_4ke.edn	EDIF netlist file
calc_4ke.ucf	User constraints file
xproj\	Project directory for the implementation tools
calc_4ke.bit	Configuration bitstream created by BitGen

Starting the Project Manager

1. Double click on the Foundation Series Project Manager icon on your desktop or select **Programs** → **Foundation Series** → **Foundation Series Project Manager** from the Start menu.

Figure 4-3 Project Manager Icon

2. If this is the first time you have run the Project Manager, it will ask if you wish to create a new project or open an existing project. Select **Open an Existing Project** and click OK.

   If this is not the first time you have run the Project Manager, select **File** → **Open Project**.

3. In the Directories list, browse to c:\fndtn\active\projects. In the Projects list, double click on calc to open it.

There are three sections in the Project Manager window: 1) the Hierarchy Browser, 2) the Project Flowchart, and 3) the Message Window.

Hierarchy Browser

The Hierarchy Browser displays the project source files in a hierarchical tree. This allows you to quickly navigate to any point in your design.

Initially, only the top-level file(s) are displayed. Next to each filename is an icon which tells you the type of file (schematic, HDL file, state machine, library, text file). If a file contains lower levels of hierarchy, the icon will have a "+" in the lower right corner. You can expand the tree by clicking on this icon.

You can open a file for editing by simply double clicking on the filename in the browser.

Project libraries are also listed in the Hierarchy Browser. The Project Manager automatically configures the appropriate Xilinx libraries based on the target device family.

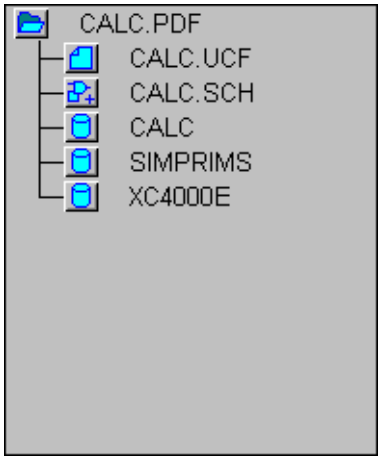


Figure 4-4 Hierarchy Browser

Project Flowchart

The Project Flowchart is a graphical view of the design process to guide you through the steps necessary to create, implement, and verify a design.

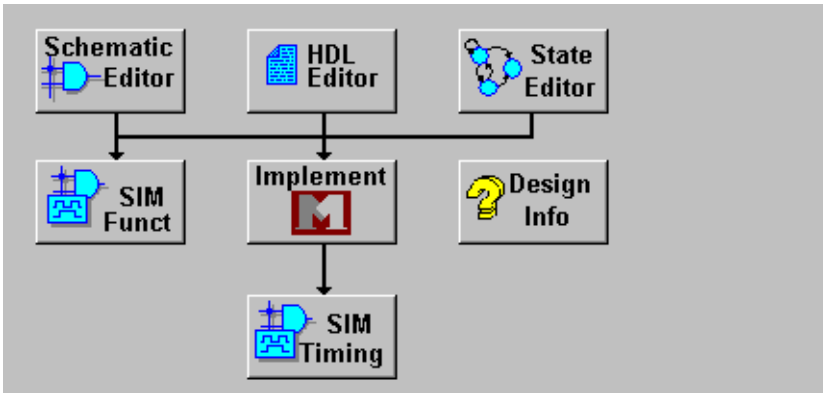


Figure 4-5 Project Flowchart

Message Window

Errors, warnings, and informational messages are displayed in the Message Window. Errors are displayed in red, warnings in blue, and informational messages in black.

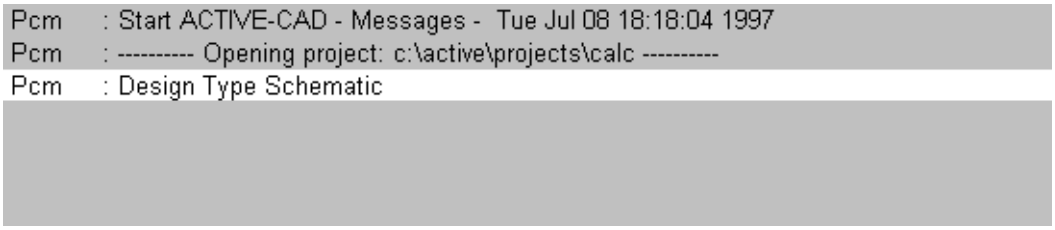


Figure 4-6 Message Window

Copying the Tutorial Files

You can either work within the calc directory as it has been installed from the CD, or you can make a copy to work on. Perform the following steps to make a working copy of the tutorial files:

1. Select **`File`** → **`Copy Project`**.
2. Under the Destination section, type “mycalc” in the Name field.
3. Click OK.
4. Select **`File`** → **`Open Project`**.

5. Scroll down in the project list and select mycalc. Click Open.
6. The mycalc project may contain two UCF files. If this is the case, select the mycalc.ucf file. Select **Document** → **Remove** or press Del to remove the file from the project. Click Yes to confirm the removal of the file.

Note: This does not delete the file from disk. If you mistakenly remove a file from a project, select **Document**→ **Add** to add it back.

Starting the Schematic Editor

To open the Calc schematic sheet in the Schematic Editor, perform the following steps:

1. Click on the Schematic Editor button in the Project Flowchart, or double click on CALC.SCH in the Hierarchy Browser.

2. The calc.sch schematic should be automatically opened. If it is not, select **File** → **Open**; select the CALC sheet; click OK.

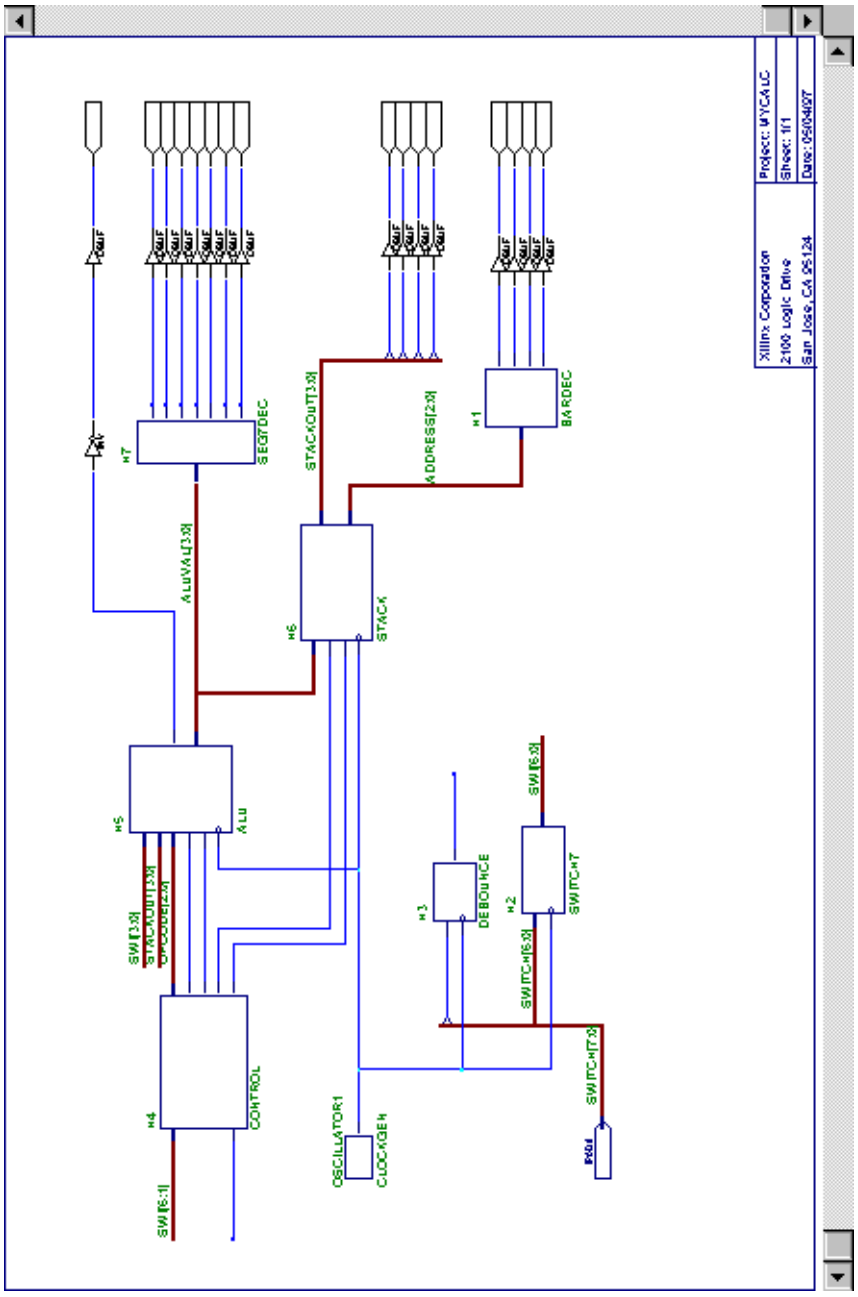


Figure 4-7 Top-level Schematic for Calc

Executing Commands

There are three ways to execute commands within the Foundation tools: pulldown menus, hotkeys, and toolbar buttons. This tutorial will instruct you to use the pulldown menus in most cases.

Hotkeys

The keyboard can be used to execute various commands. These "hotkeys" are listed next to the commands within the pulldown menus. Some of the hotkeys are the function keys, some are single letters, and some require the Ctrl or Alt keys. They are not customizable.

Toolbar Buttons

There are also two toolbars that are active around the main Schematic Editor window. Hold your mouse over the buttons to see their function.

Manipulating the Screen

Under the Display pulldown menu, you will find a series of commands that modify the viewing area of the Schematic Editor window.

Targeting an XC9500 Device

The incomplete Calc design is configured for an XC4003E-PC84 part. If you want to target a demonstration board with this device, skip this section and go directly to the "Completing the Calc Design" section. If you are targeting an XC9500 family device, you must change the project type to reference the XC9500 library instead of the XC4000E library.

The procedure provided below changes every Xilinx component in the Calc design from the XC4000E library to the XC9500 library. In the Xilinx Unified Libraries, components which are common to multiple device families have identical footprints and pinouts. This allows you to easily retarget projects to different device families, provided only library parts common to the two families are used. You must manually replace any library parts that are not common to both families. This example shows a situation where this may happen.

Note: Although an XC4000E-to-XC9500 conversion is shown here, this procedure may be used to retarget from any Xilinx family to any other Xilinx family.

To retarget the Calc project to the XC9500 family,

1. In the Schematic Editor, select **View** → **Preferences** → **General**.
2. In the General Settings dialog box, uncheck the **Add Libraries to Project** box. Click OK.

Note: Leaving this box checked may result in an invalid netlist.

3. In the Project Manager, click on the Design Info button, located in the Project Flowchart area. The Design Info dialog box appears.
4. In the Project Info section, select the XC9500 family. If desired, change the target part and speed grade.
5. Click OK, then click Yes to save the change.

Since the automatic adding of libraries was disabled in step 2, only components that have equivalents in the target technology are translated. Those components that do not have an equivalent in the target technology do not appear in the converted schematics. For example, an XC4000E RAM component has no equivalent in an XC9500 device. Therefore, if you retarget an XC4000E project with RAM components to an XC9500 device, blank spaces are left where the RAM components were placed on the original schematic. These portions of your design must be modified by hand. Refer to the "Modifying the Design for Non-XC4000 Family Devices" section for more information about how to make these edits.

Completing the Calc Design

If you need to stop the tutorial at any time, be sure to save the work you have done by first selecting **File** → **Save** from the menus.

Design Description

The top-level schematic of the Calc design has been created for you. Each of the blocks in the schematic, such as CONTROL or ALU, is linked to a second-level macro that describes its logic.

Additionally, any second-level macro can contain another block that references a third-level drawing, and so on. This organization is known as a hierarchical structure.

In this tutorial, you add some symbols to the ALU block schematic to complete it. First, you create the ANDBLK2 symbol and its underlying schematic and then add them to the schematic. Additionally, you add symbols from the Unified Libraries to the ALU block. After the ALU block is finished, you add the STARTUP block to the top-level Calc schematic to tie the device's global reset network to a device pin. To complete the design entry, you add a CONFIG block, which lists a set of instructions that dictate how the implementation tools should process the design.

Calc is a 4-bit processor with a stack. The processor performs functions between an internal register and either 1) the top of the stack or 2) data input from external switches. The results of the various operations are stored in the register and displayed in hexadecimal on a seven-segment display. The top value in the stack is displayed in binary on a bar LED. A count of the items in the stack is displayed as a "gauge" on another bar LED.

The design consists of the following functional blocks:

- ALU

  Contains the arithmetic functions of the processor.

- CONTROL

  Decodes the input opcodes into control lines for the STACK and ALU blocks.

- STACK

  A four-nibble storage device. It is implemented using synchronous RAM in an XC4000E device. The RAM4_9K macro, which uses flip-flops, can be substituted for the RAM16X4S macro in this block to implement the stack in an XC9500 or other non-XC4000 family device.

- DEBOUNCE

  Debounces the "execute" switch, providing a one-shot output.

- SEG7DEC

  Decodes the output of the ALU for display on the 7-segment decoder.

- CLOCKGEN

  Uses an internal oscillator circuit in XC4000 family devices to generate the clock signal. When targeting an XC9500 device, it is replaced by an input pad and a clock buffer.

- BARDEC

  Shows how many items are on the stack on a "gauge" of four LEDs.

- SWITCH7

  Latches the input coming from the switches on the demonstration board.

Creating the ANDBLK2 Symbol

1. Select **`Hierarchy`** → **`New Symbol Wizard`**. The Design Wizard appears.

   The Design Wizard guides you through the process of creating a macro symbol. It also creates a "skeleton" file based on the pins defined and the type of macro (schematic, ABEL, VHDL, or state machine). The State Editor and the HDL Editor (described later in this tutorial) also use the Design Wizard.

2. Click Next.

3. In the Symbol Name field, type "andblk2". In the Contents section, select Schematic.

4. Click Next.

5. Click New to create a new pin. In the Name field, type "a[3:0]".

Note: You can also type "a" in this field and use the arrow buttons in the Bus section to select the bus indices.

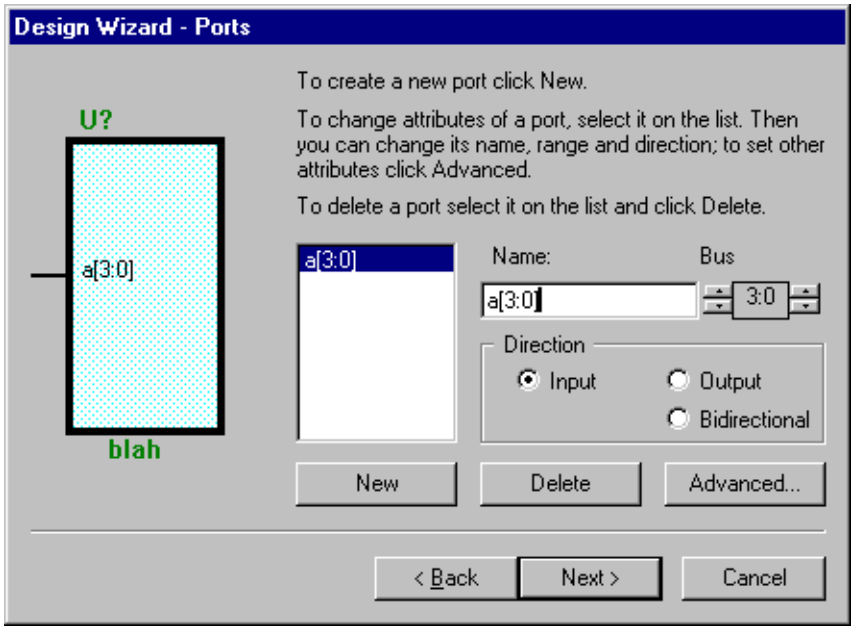


Figure 4-8 ANDBLK2 Symbol After Adding Pin a[3:0]

6. Repeat step 5 for input pin b[3:0].
7. Repeat step 5 for q[3:0], but this time select Output in the Direction section.

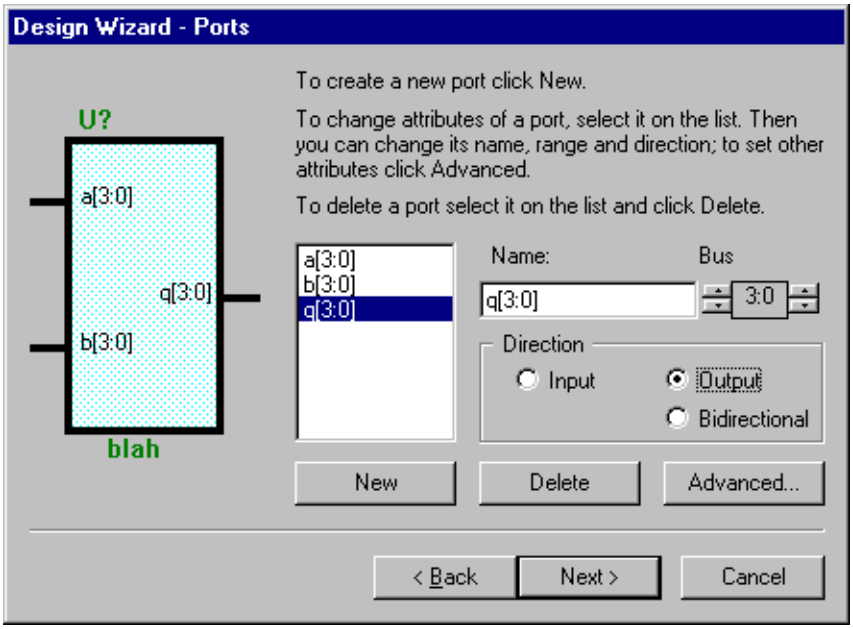


Figure 4-9 Completed ANDBLK2 Symbol

8. Click Next.

Note: In the Comments section, you can type text that will appear on the symbol when it is placed. You can also define a longer comment that only appears in the SC Symbols Window when you are placing components.

9. Click Next, then click Finish.

   The Symbol Wizard automatically creates and opens a schematic sheet with I/O terminals corresponding to the defined symbol pins.

Note: If the schematic is not automatically created, the most likely cause is that Empty was selected in step 4. Repeat steps 1-9, and click Yes or OK when prompted to overwrite the existing symbol.

Creating the ANDBLK2 Schematic

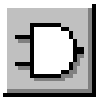
You have created the symbol for ANDBLK2. The next step is to create a schematic for this macro. The schematic can then be referenced in a higher-level schematic by placing the symbol.

Opening the Schematic

1. If the ANDBLK2 schematic is not open, select **File** → **Open**. The Open Sheet dialog box appears.
2. Click Browse, select andblk2.sch from the files list, then click OK. Zoom in until all of the I/O terminals are clearly visible.

Adding Components

1. From the menu bar, select **Mode** → **Symbols** or click on the Symbols Toolbox button in the vertical toolbar. This opens the SC Symbols window. The SC Symbols window lists available components from all of the libraries associated with this project.



2. Select AND2. You can also type “AND2” in the bottom of the SC Symbols Window. Then move the mouse back into the schematic window.
3. Move the symbol outline to the location shown in the following figure and click the left mouse button to place the object.

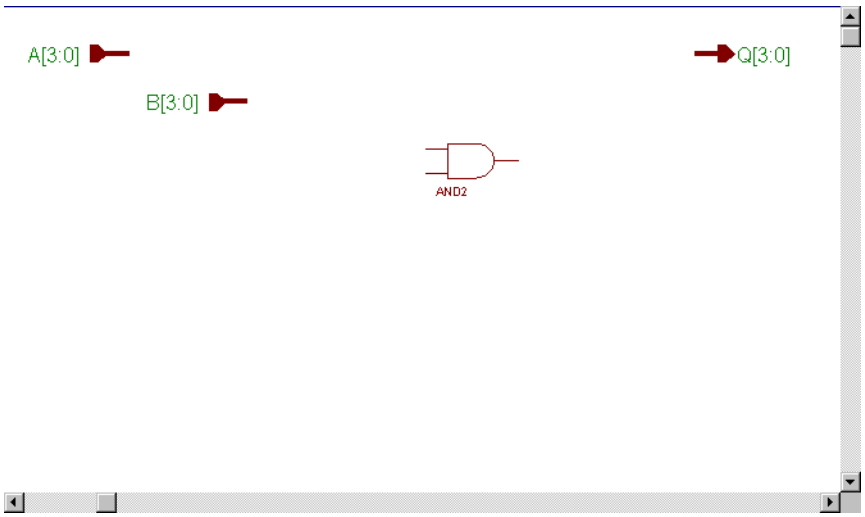


Figure 4-10 Placing a Component

Placing Additional Components

Place more components of the same type by clicking three times on the AND2 component (the same component that you have already placed once) and then moving each component to the desired location. When completed, close the SC Symbols window to exit the Symbols mode.

Moving a Component

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

1. Press Esc to exit the Symbols Mode.
2. Select the component to be moved. Make sure that no other components are selected (clicking on a blank area of the schematic unselects everything).
3. Click and drag to correctly place the component.

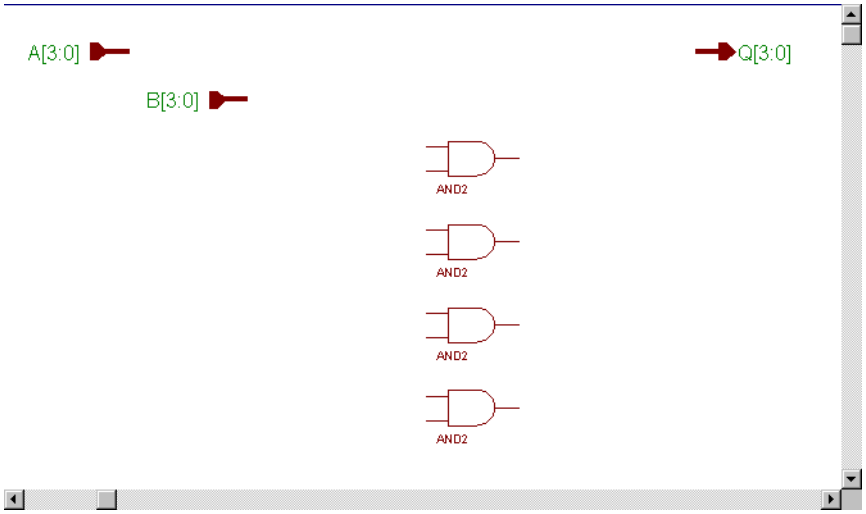


Figure 4-11 Component Placements for ANDBLK2

Adding Buses

Sometimes it is convenient to draw a set of signals as a bus rather than as several separate wires.

Add buses to the schematic as follows:

1. Select **Mode** → **Draw Buses** or click on the Draw Buses button in the vertical toolbar.

2. The ANDBLK2 schematic has some bus "stubs" connected to I/O terminals which represent the symbol pins as defined with the Symbol Wizard. Click on the end of the stub, then move the mouse to a new position. Click to make a corner in the bus.
3. Terminate the bus by double clicking. This opens a dialog box where you can define the bus name, width, and the type of I/O terminal used.

4. Click Bus End (the bus name and I/O terminal were defined with the Symbol Wizard). Add the three buses shown in the figure below.

   If you make a mistake, press Esc to exit the Draw Buses mode. Then click on the bus you want to delete so that it is highlighted. Press Del to remove the bus.

5. After adding the three buses shown below, press Esc or right-click to exit the Draw Buses mode.

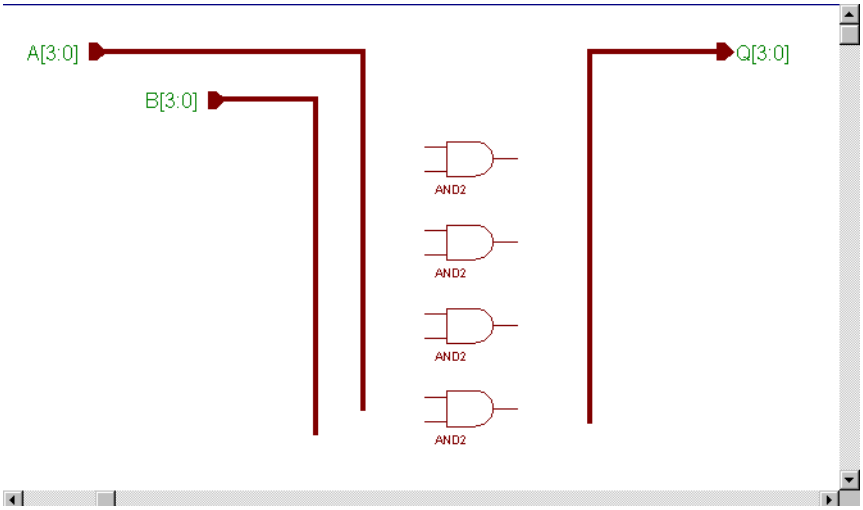


Figure 4-12 ANDBLK2 Schematic with Buses

Adding Bus Taps

Next, nets must be added to attach the appropriate pins on the AND2 components to the buses. Nets that connect a bus to another component are called bus taps. The Schematic Editor can automatically name the bus taps as they are drawn.

You may want to enlarge the view of the schematic to make it easier to draw the nets.

1. Select **Mode** → **Draw Bus Taps** or click on the Draw Bus Taps button in the vertical toolbar.

2. Click on the A[3:0] bus.

   The status bar at the bottom of the window displays the message `Expand Bus Tap: A0`. This tells you that the next bus tap drawn will be labeled A0.

Note: The default is to start at 0 and increment as bus taps are drawn. You can use the up and down arrow keys to change which bus bit will be tapped. If the last key you press is the down arrow, the bus taps will decrement as they are drawn.

3. Click on the top input pin of the top AND2 component to draw the bus tap. The status bar now reads `Expand Bus Tap: A1`.

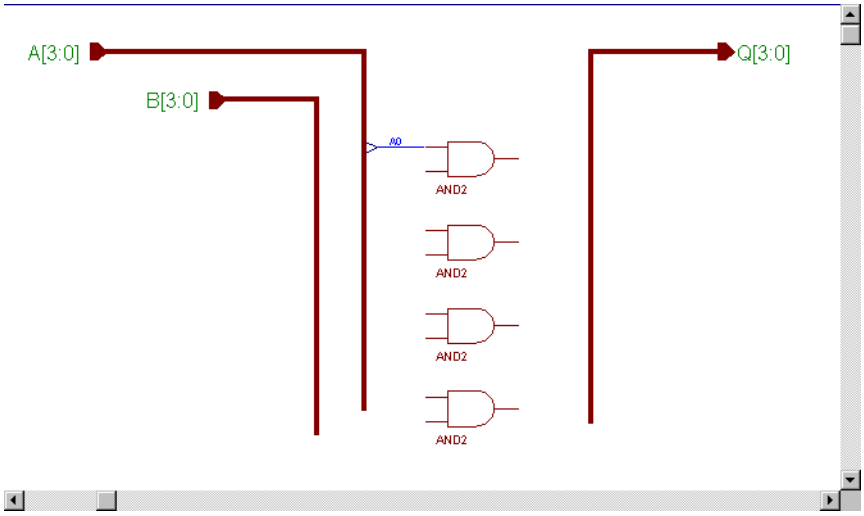


Figure 4-13 Drawing a Bus Tap

4. Click on the top input pins of the other AND2 components to place bus taps for A1 through A3.
5. Click on the B[3:0] bus. Now the status bar reads `Expand Bus Tap: B0`.
6. Add the remaining bus taps as shown in the figure below. Press Esc twice or right-click to exit the Draw Bus Taps mode.

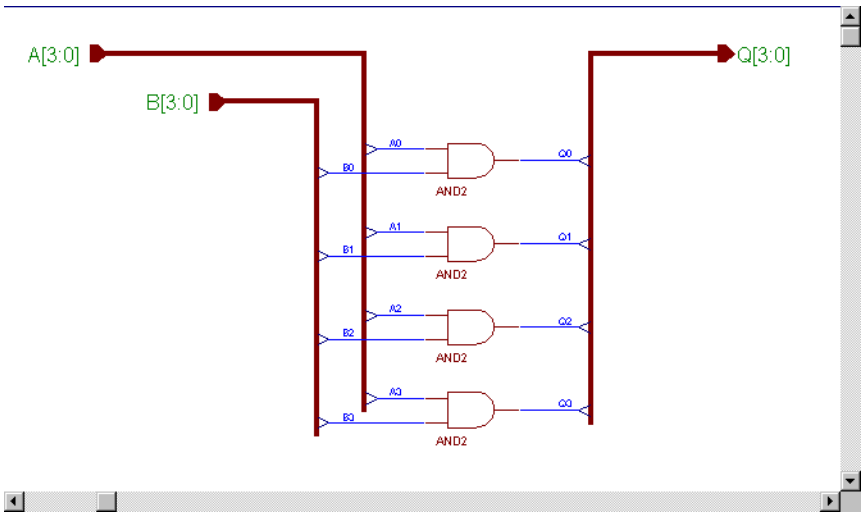


Figure 4-14 Completed ANDBLK2 Schematic

Connectivity

The logical connection between the symbol and its underlying schematic is done through the use of I/O terminals. The name of each pin on the symbol must have a corresponding terminal in the underlying schematic.

The Symbol Wizard automatically places I/O terminals on the schematic. You can add I/O terminals by clicking on the I/O Terminal button in the vertical toolbar. When you save a macro, the Schematic Editor checks the I/O terminals against the corresponding symbol. If there is a discrepancy, you can let the software update the symbol automatically, or you can modify the symbol manually.

Saving the Schematic

The schematic is now complete.

1. Save the schematic by selecting **File** → **Save** or clicking on 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.

2. Select **File** → **Close**.

Completing the ALU Schematic

So far you have created the ANDBLK2 macro. The next step is to complete the ALU schematic.

1. If the CALC schematic is not open, select **File** → **Open**, select the CALC sheet, and click OK.
2. Select **Hierarchy** → **Hierarchy Push** or click on the Hierarchy Push / Pop button in the vertical toolbar.

3. Double click on the ALU symbol. This will push into the schematic below the symbol.
4. Press Esc or right-click to exit the Hierarchy Push / Pop mode.

Placing User-created Components

The ANDBLK2 symbol can now be placed on the schematic as shown in the figure below. Place the symbol using the same procedure you used to place the AND2 gate when you created the ANDBLK2 schematic.

1. Use the Display menu commands to zoom into the empty area near the center of the schematic, below the ORBLK2 symbol.
2. Click on the Symbols Toolbox button.
3. Scroll down and select the ANDBLK2 component from the SC Symbols window and place it in the empty area.

4. Press Esc to exit the Symbols mode.

   Notice that the SC Symbols window remains open. This allows you to quickly place additional symbols without having to click on the Symbols Toolbox icon again.

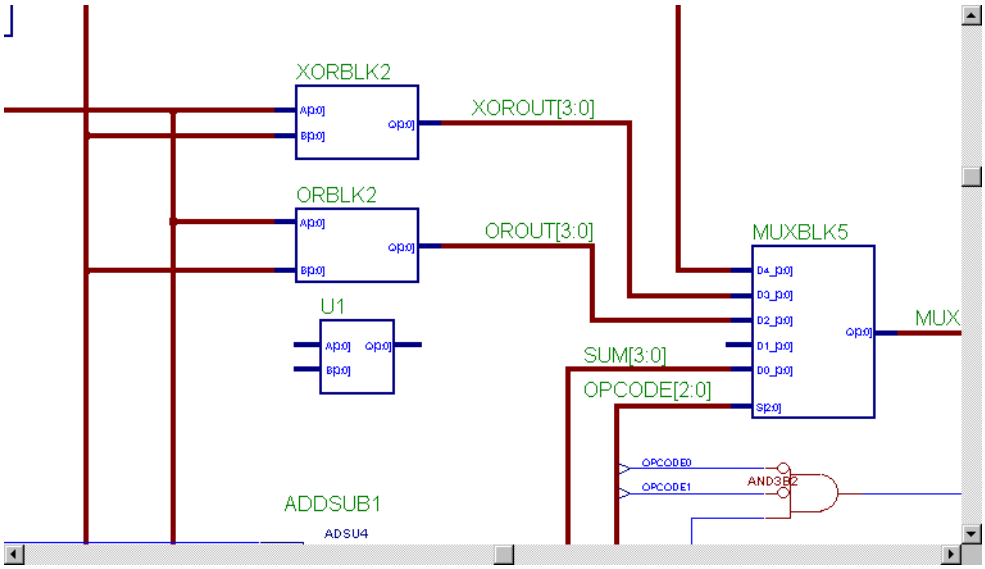


Figure 4-15 Adding ANDBLK2 to ALU Schematic

Placing Library Components

The next step is to add the FD4RE and AND5B2 components to the ALU schematic. Both of these components are available in the Xilinx Unified Libraries. The FD4RE component consists of four flip-flops with clock enables. The AND5B2 component is a five-input AND gate with two inputs inverted ("bubbled," hence the "B").

1. Zoom into the open area in the lower right corner.
2. Select the FD4RE component by using the scroll bar or by typing the name in the SC Symbols window.
3. Place the component in the lower right corner of the schematic, according to the figure below.
4. Repeat steps 2 and 3 to place an AND5B2 component, as shown in the figure below.
5. Close the SC Symbols window.

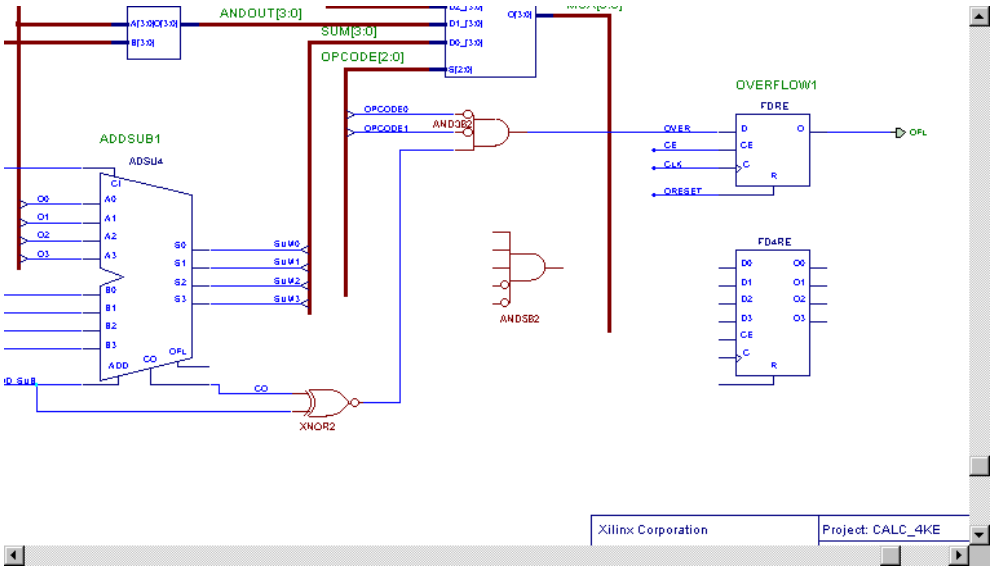


Figure 4-16 Adding FD4RE and AND5B2 to ALU Schematic

Adding Nets, Buses, and Labels

Next complete the addition of the FD4RE and AND5B2 symbols by adding buses, bus taps, nets, and labels as follows:

1. Add the necessary buses, bus taps, and nets to complete connections for FD4RE and AND5B2 as you did for the ANDBLK2 schematic. The figure below displays the labeled nets and buses for FD4RE and AND5B2. To add a net, select **Mode** → **Draw Wires** or click on the Draw Wires button in the vertical toolbar. Draw nets in the same way as you draw buses (double click to create a hanging end).

2. To add labels to nets, double click on the net. Type the label in the Net Name field. The nets should be labeled as shown.
3. When you end a bus by double clicking (to create a hanging stub), a dialog box appears. Enter the label in the Bus Name field, and in the I/O Marker list box, select None.

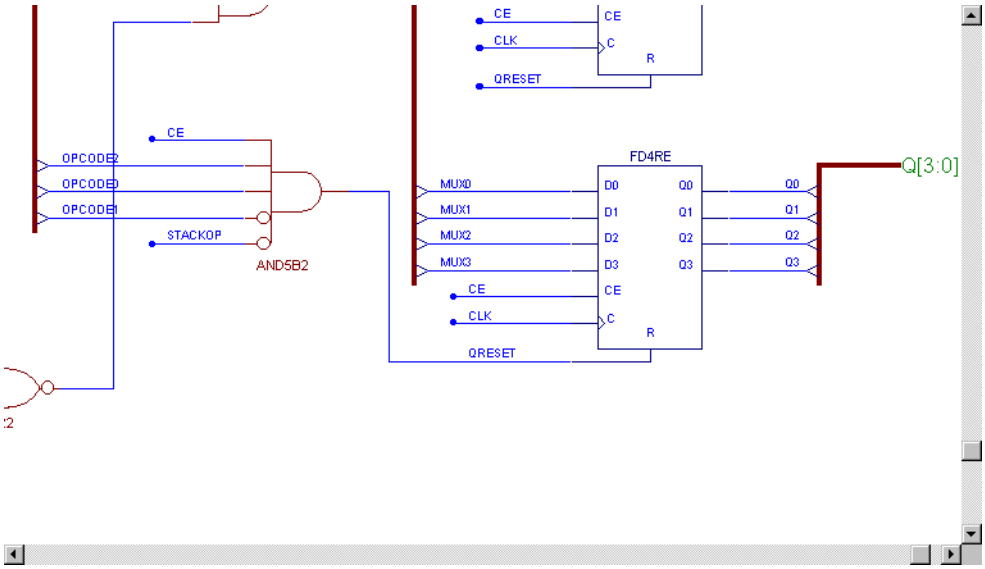


Figure 4-17 Nets, Buses and Labels for FD4RE and AND5B2

Next, complete the addition of ANDBLK2 to the ALU schematic.

4. Add the buses shown in the figure below to connect ANDBLK2. Add a label to the output bus in the same way that net labels are

added. The figure below displays the labeled nets and buses for ANDBLK2.

Note: You may want to reposition the ANDBLK2 symbol to make the buses neater. If you move a component, any nets or buses connected to it will move as well. This indicates that the connections have been made successfully.

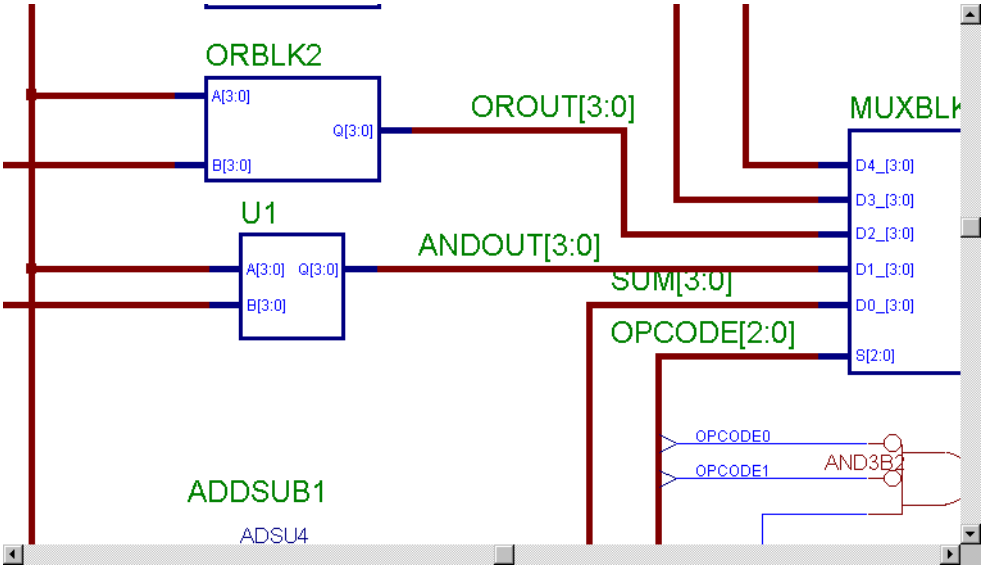


Figure 4-18 Bus and Label for ANDBLK2

Changing Symbol References

It is important to add references to symbols. Error and warning messages often specify symbol references, and references also appear in simulation netlists. Also, net names at lower levels of hierarchy are specified using the following format:

```
symbol_reference1/symbol_reference2/net_label
```

In the ALU schematic, references have already been added to the MUXBLK2, XORBLK2, ORBLK2, and MUXBLK5 blocks.

To add a reference to the ANDBLK2 component, follow these steps.

1. Double click on the ANDBLK2 symbol. The Symbol Properties dialog box appears.

2. In the Reference field, type ANDBLK2; click Apply. Click OK.

Note: Foundation requires that all references end with a number.

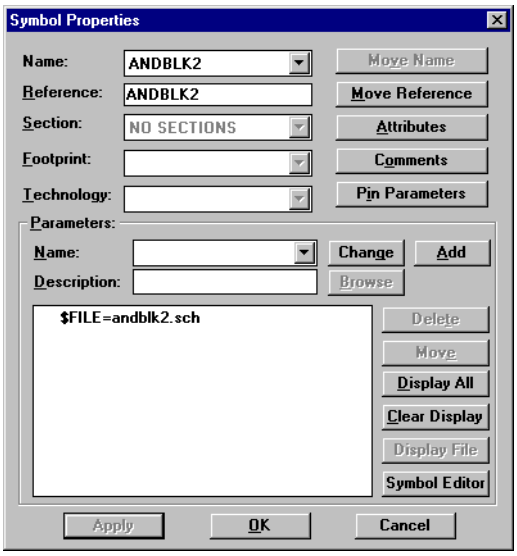


Figure 4-19 Adding Symbol References to ALU Schematic

3. Give the FD4RE component the reference ALUVAL1.

The completed ALU schematic is shown in the following figure.

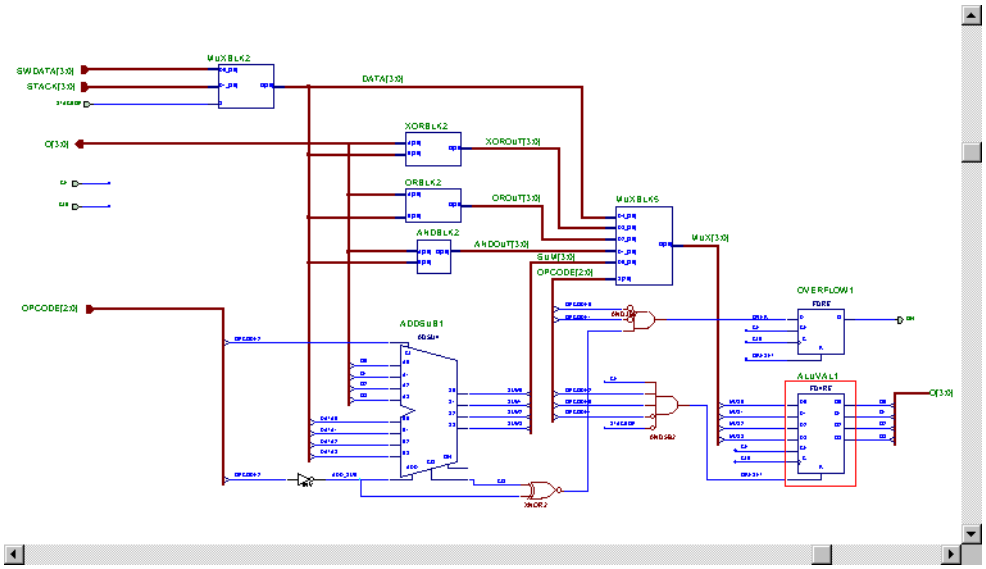


Figure 4-20 Completed ALU Schematic

Saving the ALU Schematic

Save the schematic. If errors occur, resolve them and then save the schematic again.

Exploring Xilinx Library Elements

The Xilinx libraries contain three types of elements. Primitives are basic logic elements, such as the AND2 gates that you previously placed in ANDBLK2. Soft macros are schematics created by combining primitives and other soft macros. Relationally Placed Macros (RPMs) are soft macros that contain placement information. RPMs are currently only available in the XC4000 family libraries.

All three types of library elements are placed on a schematic in exactly the same way.

Viewing a Xilinx Soft Macro

Soft macro schematics are similar to schematics you create for your own projects.

Open the schematic underneath the FD4RE symbol as follows:

1. Click on the Hierarchy Push / Pop button.
2. Double click on the FD4RE symbol.

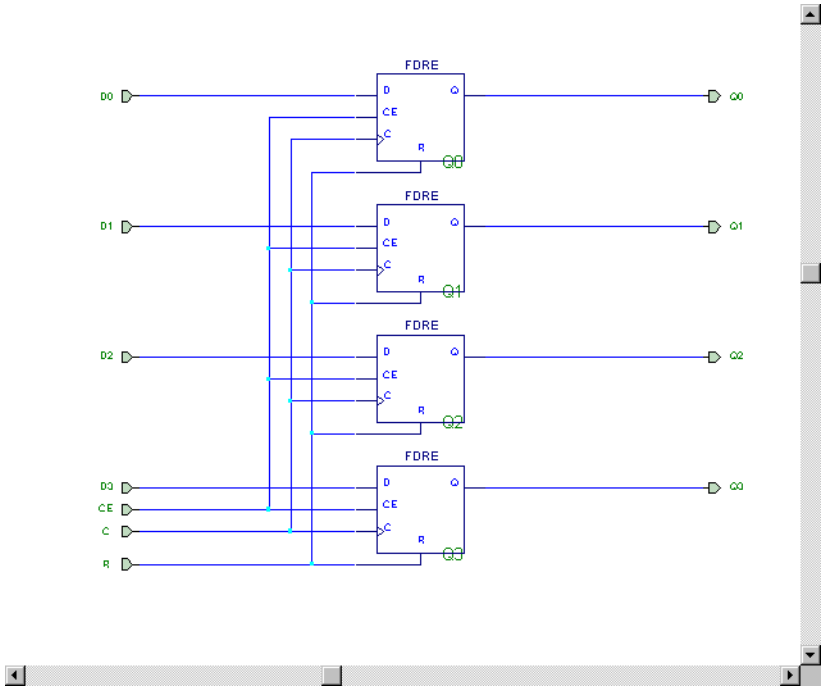


Figure 4-21 FD4RE Schematic from XC4000E Library

3. Select **Hierarchy** → **Hierarchy Pop** or double click on an empty space in the schematic to return to the ALU schematic.

Viewing a Xilinx RPM (XC4000 Family Only)

Note: The following description of RPMs contains detailed information on the XC4000E architecture. Refer to *The Programmable Logic Data Book* for more information on the XC4000E CLB structure and fast carry logic.

If your project is not targeted for an XC4000E device, read this section, but do not perform any of the commands. Continue the tutorial with the next section, the "Returning to the Calc Schematic" section.

The ALU contains a component from the Xilinx library, ADSU4, which is a four-bit wide adder/subtracter. If your project is targeted for an XC4000E device, this schematic is implemented as a Relationally Placed Macro (RPM). If your project is targeted for an XC9500 device, the ADSU4 is implemented without this placement information.

RPM schematics are similar to schematics you create for your own projects. You can load one of these schematics and use the **File** → **Save As** command to save it under another name. You can then edit this new schematic to customize it to your needs.

Elements placed in the ADSU4 RPM schematic include CY4 components and FMAPs. The CY4 symbol specifies fast carry logic functionality from the schematic. Fast carry logic is a hardware feature in XC4000 family parts that allows very fast arithmetic functions.

The FMAPs map logic functions into function generators within the configurable logic blocks (CLBs), which are arranged in a rectangular grid in the die. Both the CY4 symbols and FMAP symbols have RLOC parameters on them. RLOCs assign relative CLB locations to the components. You can use carry symbols as well as FMAPs and other mapping components to create RPMs. However, knowledge of them is not necessary to use RPMs. Only expert users should create new macros containing carry logic. For a description of these components, see the "Attributes, Constraints, and Carry Logic" chapter in the DynaText online *Libraries Guide*.

Push into the ADSU4 schematic as follows:

1. Click on the Hierarchy Push / Pop button.
2. Double click on the ADSU4 symbol. Press Esc or right-click to exit the Hierarchy Push / Pop mode.
3. Zoom into the upper portion of the schematic as shown in the following figure.

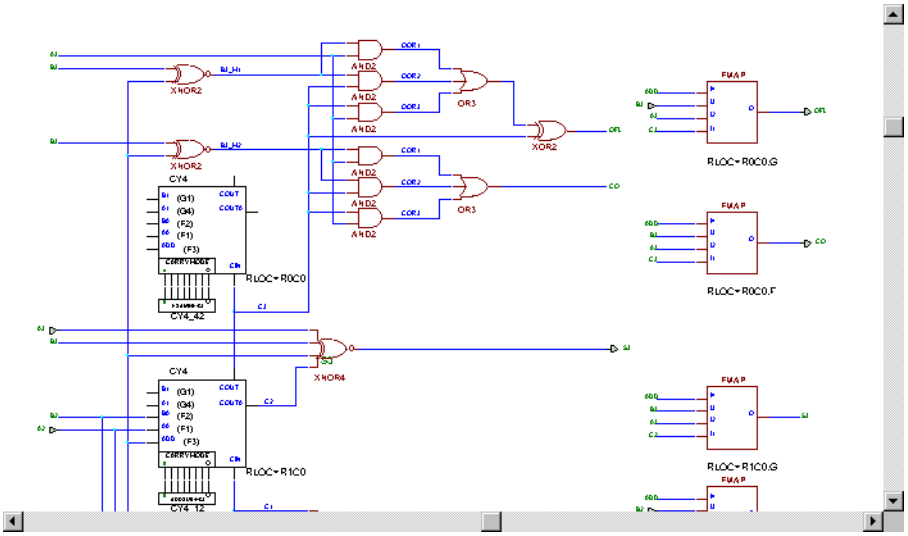


Figure 4-22 Upper Portion of the ADSU4 RPM Schematic

4. Double click on the FMAP symbol in the upper right corner. The Symbol Properties dialog box appears.

   The RLOC parameter is set to R0C0.G, indicating that this function is mapped to the G function generator of the upper-leftmost (row zero, column zero) CLB in the RPM. RPM origins are in the upper left corner.

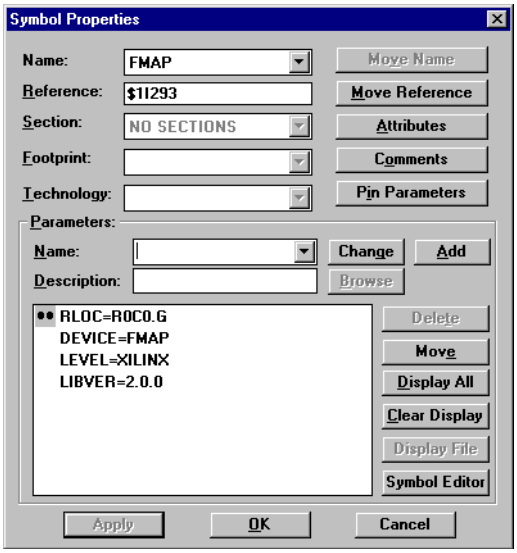


Figure 4-23 RLOC Parameter on FMAP Component

5. Click OK to return to the ADSU4 schematic window.
6. Use the scroll bars on the sides of the window to pan around the schematic and look at the RLOCs.

   The logic is mapped to three CLBs, designated as R0C0, R1C0, and R2C0. Therefore, this RPM uses three CLBs that are arranged in a column. The number of CLBs used and the shape of each RPM is documented in the online *Libraries Guide*. Note that these locations are relative, not absolute. Regardless of the RPMs absolute location inside the target device, the logic associated with the FMAP with the location R0C0 is always at the top, R1C1 is in the CLB directly below, and so on.
7. Select **`Hierarchy`** → **`Hierarchy Pop`** to return to the ALU schematic.

Returning to the Calc Schematic

1. Select **`Hierarchy`** → **`Hierarchy Pop`** to return to the top level Calc schematic.
2. Press Esc or right-click to exit the Hierarchy Push / Pop mode.

Using the XC4000E Oscillator

If your project is not targeted for the XC4000 family, read this section, but do not perform any of the commands.

The XC4000 family devices contain an on-chip oscillator. The frequency of this oscillator is not precise, but is suitable for designs that do not need a highly accurate clock, such as Calc.

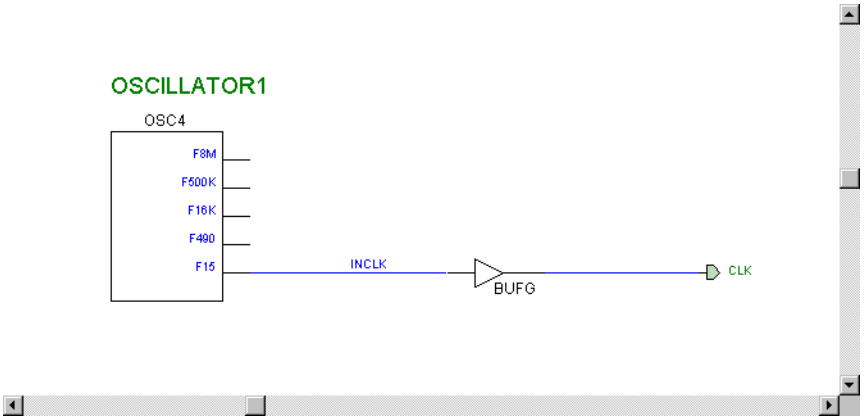


Figure 4-24 CLOCKGEN Schematic

The CLOCKGEN schematic contains an XC4000E library part, OSC4. This symbol represents the on-chip oscillator that generates nominal clock frequencies of 8 MHz, 500 kHz, 16 kHz, 490 Hz, and 15 Hz. Calc uses the 15 Hz output from this component when targeted for XC4000 family devices. The clock output from OSC4 is buffered through a BUFG global clock buffer.

XC4000E devices have eight on-chip clock buffers: one BUFGP (primary global buffer) and one BUFGS (secondary global buffer) in each corner of the device. Although it is possible to use them for other purposes, BUFGPs are best used to route externally-generated clock signals. BUFGSs have more flexibility and can be used to route any large fan-out net, even if it is internally sourced. A BUFG symbol can represent either type of buffer and allows the implementation software to choose which type of global buffer is best. Using a BUFG also facilitates retargeting to other Xilinx device families, since it can represent any type of global buffer in any family. The BUFG in the Calc schematic is substituted for a BUFGS during implementation, because the clock is generated internally by the on-chip oscillator.

See the online *Libraries Guide* and *The Programmable Logic Data Book* for more information on global clock buffers for Xilinx devices.

Controlling Implementation from the Schematic

The following subsections explain how to control design implementation within your schematic.

Assigning Pin Locations (XC4000 Family Only)

Xilinx recommends that you let the automatic placement and routing program, PAR, define the pinout of your design. Pre-assigning locations to the pins can sometimes degrade the performance of the place-and-route tools. However, it is usually necessary, at some point, to lock the pinout of a design so that it can be integrated into a PCB (printed circuit board).

The initial pinout should be defined by running the place-and-route tools without pin assignments, then locking down the pin placement so that it reflects the locations chosen by the tools. The pins in the Calc design must be assigned locations so that it can function in a Xilinx demonstration board. Because the design is simple and timing is not critical, these pin assignments will not adversely affect the ability of PAR to place-and-route the design.

Pin locations are specified by attaching a LOC parameter to a pad component. Assign a LOC parameter to the pad associated with the STACKLED0 signal on the Calc schematic as follows:

1. Double click on the OPAD connected to the net labeled STACKLED0. The Symbol Properties dialog box appears.
2. In the Parameters section, add a new parameter with these values:

   Name: LOC

   Description: P60

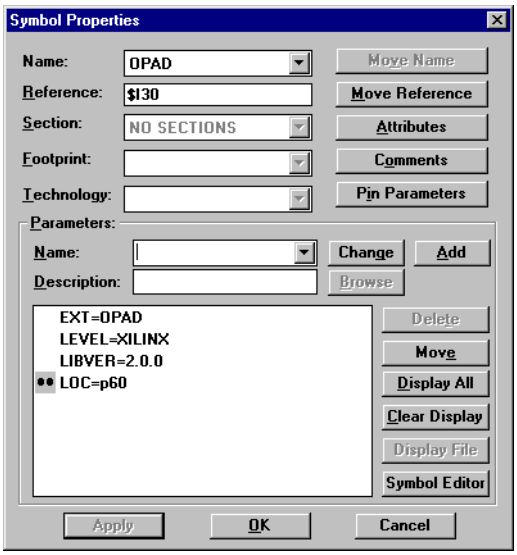


Figure 4-25 Assigning a Location to a Pad

3. Click Add. The parameter appears in the list box.
4. Notice the single black dot to the left of the parameter. This indicates that only the Description field will be displayed on the schematic. Double click on the parameter until two dots are shown. This indicates that both the Name and Description fields will be displayed on the schematic.
5. Click Apply. You will see the parameter, next to the OPAD.
6. Click OK to close the window.

The other pin locations for the Calc design have been placed in a data file known as a constraint file.

Valid pin locations vary depending on the package. PLCC, HQFP, and other “numeric-only” package pins are designated with a P followed by the pin number, such as P17. PGA and other grid-array package pins use alphanumerics such as A12. *The Programmable Logic Data Book* lists the pinouts of each FPGA and CPLD for each package that Xilinx supplies.

Controlling Slew Rate

Output slew rate can be modified by assigning a FAST parameter (constraint) to an output pad, as shown in the following figure. The default slew rate is SLOW. “Fast” pads have different timing specifications and draw more current than “slow” (slew-rate-limited) pads. Slow pads are used by default. See *The Programmable Logic Data Book* for timing specifications for the various slew rate modes.

Add the FAST parameter to the STACKLED[3:0] pads as follows:

1. Double click on the OPAD connected to the net labeled STACKLED0.
2. In the Parameters section, add a new parameter with these values:

   Name: FAST

   Leave the Description field blank.
3. Click Add. Double click on the parameter in the list until two black dots are displayed.

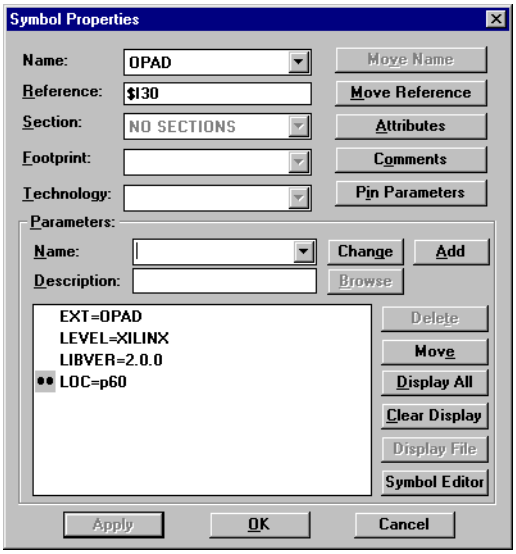


Figure 4-26 Designating a FAST Pad

4. Click OK. You will see the parameter next to the OPAD.

   The parameter may be displayed on top of the other OPADs. To move a symbol's parameters, double click on the symbol. In the parameter Parameters section, click Move. Use the mouse to position the parameters and click the left mouse button to place them.

5. Repeat steps 1 through 4 for the remaining OPADs connected to the STACKLED nets.

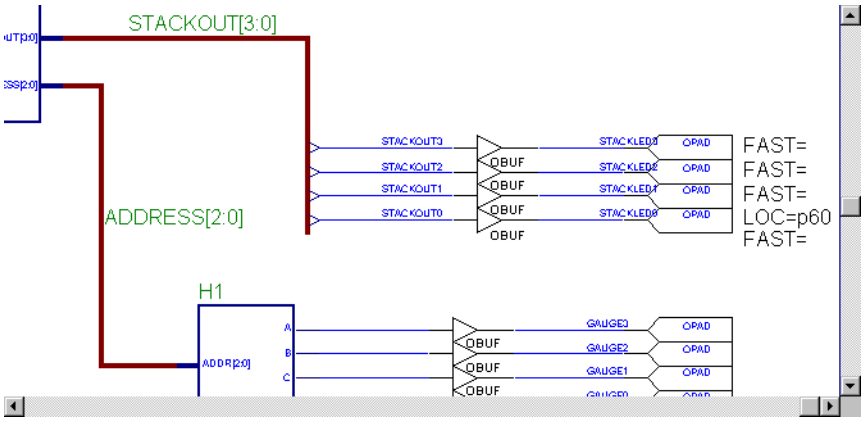


Figure 4-27 LOC and FAST Parameters on STACKLED Pads

Using I/O Flip-flops

Xilinx XC4000 family devices have two flip-flops in each I/O block (IOB). Each pad has an associated input flip-flop and output flip-flop. You can also configure input flip-flops as level-sensitive latches and output flip-flops can be 3-stated. You access these elements using the library components IFD, ILD, OFD, and OFDT, as well as other higher-level macros that contain these components. For more information on these library elements, consult the online *Libraries Guide*.

IOB flip-flops can be used to free up internal CLB resources. In this design, IOB flip-flops are used to register the switch inputs. As shown in the figure below, the SWITCH7 macro attached to the input bus SW[7:0] in the lower left area of the schematic has an underlying schematic that consists of seven IFDs.

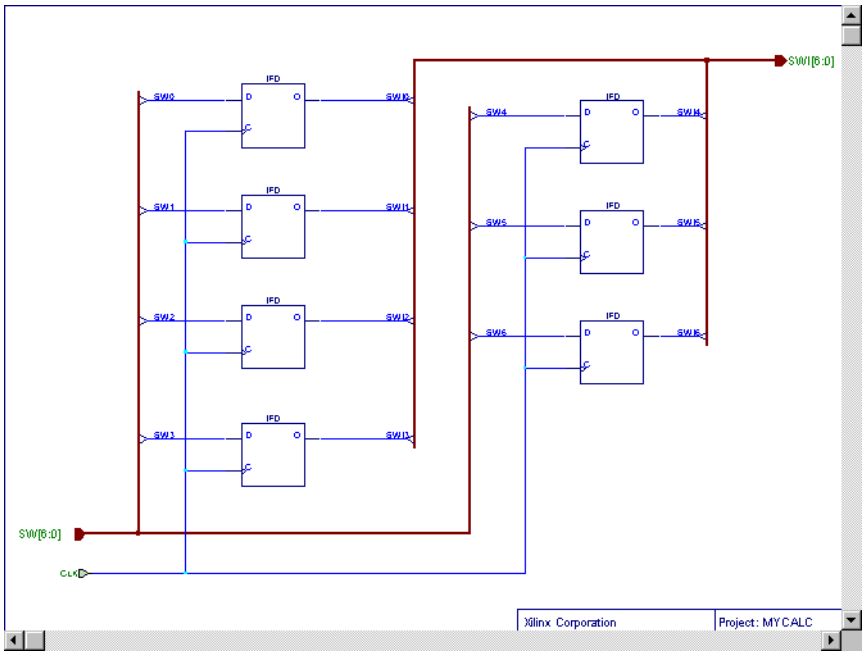


Figure 4-28 SWITCH7 Schematic Using Input Flip-Flops

Saving the Calc Schematic

Before continuing, save the changes made to Calc.

Modifying the Design for Non-XC4000 Family Devices

At this point in the tutorial, you have created or edited the following schematic files: CALC, ALU, and ANDBLK2. The design is currently suitable for use only in an XC4000 family device. This is because these devices have several features not found in other Xilinx device families. Two of these features are the on-chip memory built into the XC4000 CLB and the on-chip oscillator.

RAM Stack Implementation

The RAM stack is implemented using a 16x4 RAM component from the XC4000E library, which occupies two CLBs.

Although the stack is only 4x4, memory components are only available in 16x1 or 32x1 increments, so only one fourth of the memory addresses are used. A stack four times as deep could be implemented in the same two CLBs. An equivalent flip-flop implementation of the stack would require 16 flip-flops or 8 CLBs.

A device-independent stack has been implemented by replacing the RAM16X4S with a register file that emulates a synchronous RAM with a set of flip-flops and multiplexers. This implementation can be used for any Xilinx device.

Note: If you are targeting an XC4000 family device, skip this section.

Make the stack a device-independent schematic as follows:

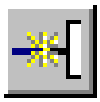
1. Push into the STACK symbol.

   If you turned off the Add Libraries to Project option, a message appears saying:

   ```
   There are missing symbol in [STACK] schematic.
   ATTENTION!
   The automatic adding of libraries to the project is disabled.
   Do you want to continue loading?
   ```

2. Click Yes to load the schematic. The empty space in the upper right corner is where the RAM component used to be.
3. Click on the Symbols Toolbox button and select the RAM4_9K component.
4. Place the RAM4_9K symbol in the empty space where the RAM16X4S symbol was. You may need to adjust the symbol so that the hanging wires meet up with the symbol pins.
5. Close the SC Symbols window.
6. Select the RAM4_9K symbol. Place the symbol.

7. Click on the Connect Symbol button in the horizontal toolbar. This connects the hanging wires to the symbol pins. Verify that the connections were made by clicking and dragging the RAM4_9K symbol. All of the connected nets will move with the symbol. If any nets are not connected, select the end of the hanging wire and drag it over the appropriate symbol pin.



8. The A3 pin on the RAM16X4S does not exist on the RAM4_9K symbol. Although the detached GND symbol and net are trimmed during the implementation process, you can clean up the schematic by deleting them. Select the GND symbol, then press Del to delete the component. The net connected to the GND will be deleted as well.
9. Save the updated STACK schematic. You may be prompted with the message:

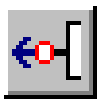
   ```
   Schematic [STACK] has been automatically corrected!
   Do you wish to continue updating this file?
   ```

10. Click Yes to save the schematic.

Removing the XC4000E Oscillator

If you are targeting the Calc design to an XC9500 or other device outside the XC4000 family, you must also remove the CLOCKGEN circuitry, which includes the OSC4 component, and replace it with an external source.

1. Select the CLOCKGEN component in calc.sch, then click on the Disconnect Symbol button in the horizontal toolbar.



2. Press Del to delete the component.
3. Add components, nets, and labels as shown below.
4. Save the Calc schematic.

Since the CLK signal is now sourced by a pad, it must be generated externally.

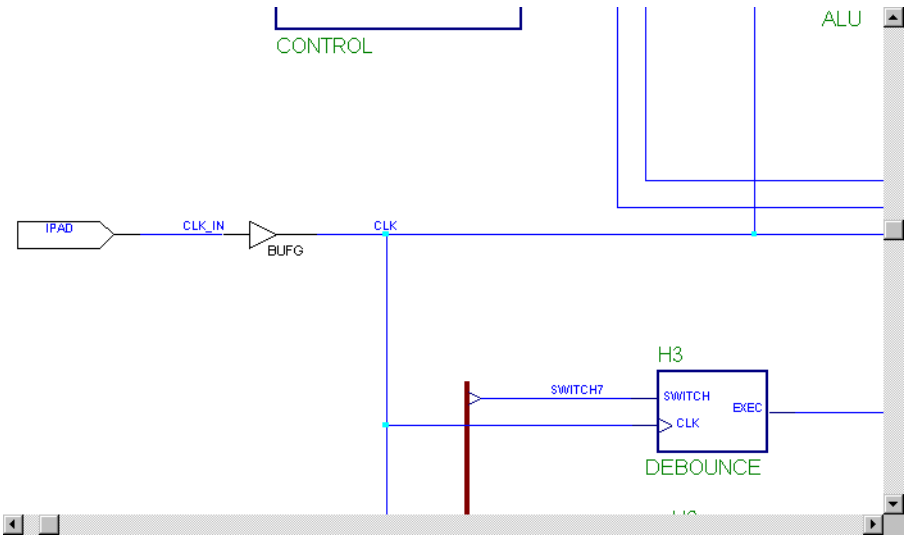


Figure 4-29 Device-Independent Clock Source

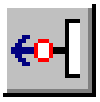
Using LogiBLOX (Optional)

LogiBLOX is a tool that allows you to quickly synthesize modules for common functions such as adders, counters, and multiplexers. It allows you to create components of any bus width (e.g., a 17-bit adder) and automatically uses the best architectural resources for a particular target device. In this section, the ADSU4 component in the ALU schematic is replaced with a LogiBLOX adder.

Note: LogiBLOX is only supported in Foundation Standard and Standard-VHDL packages and does not support CPLDs.

To replace the ADSU4 symbol with a LogiBLOX module,

1. Push into the ALU symbol and select the ADSU4 component.
2. Click on the Disconnect Symbol button in the horizontal toolbar.



3. Press Del to delete the component.
4. In the Project Manager, select **Tools** → **LogiBLOX** or from the Schematic Editor, select **Options** → **LogiBLOX**. The LogiBLOX Module Selector dialog box appears.
5. In the Module Name field, type addsub4. This will be the component name for the new LogiBLOX module.
6. In the Module Type field, use the pulldown tab to select Adders/Subtracters. The symbol will be updated to show this type of component. The Bus Width field should show 4.
7. In the Operation field, use the pulldown tab to select Add/Subtract.
8. Single click with the left mouse button to place a check mark in the Carry Input, Carry Output, and Sum boxes. Deselect any other checkboxes. The LogiBLOX Module Selector dialog box should resemble the following figure.

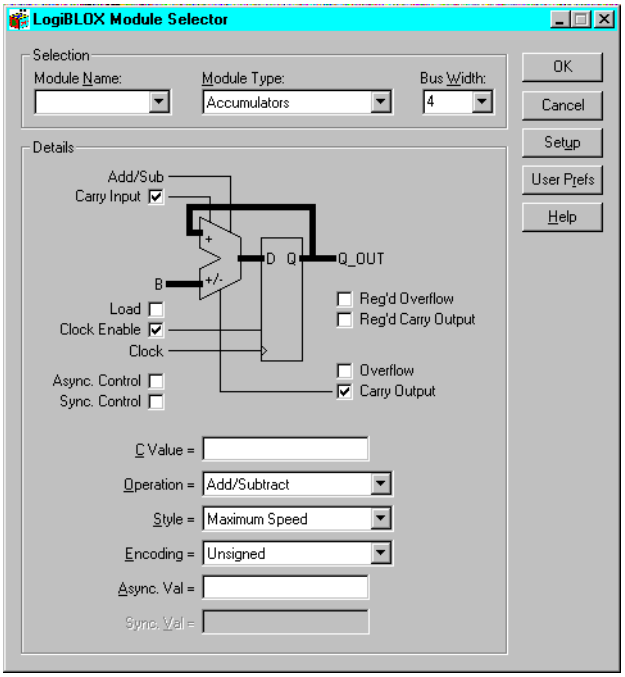


Figure 4-30 LogiBLOX Module Selector Dialog Box

9. Click on OK. A symbol and an EDIF model will be created for this LogiBLOX component. This will take a few moments.
10. After the LogiBLOX symbol has been created, place it in the space left by the ADSU4. Do not worry about lining up pins with nets right now.
11. Complete the schematic by connecting the existing nets and buses. To move a net or bus, click and drag the end of the net or bus.

    The locations of some pins are different with the LogiBLOX component, as well as the fact that data inputs and output are now bus-wide. Your schematic should resemble the following figure.

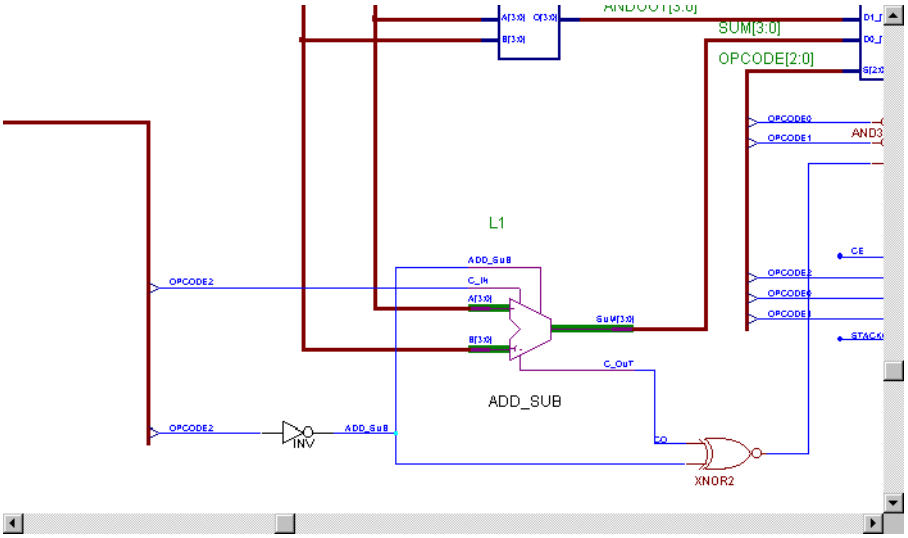


Figure 4-31 Completed ALU Schematic with LogiBLOX

12. Save the ALU schematic.

Using the State Editor (Optional)

Note: To use the State Editor, you must have either XABEL or X-VHDL installed.

The State Editor is a tool that converts a graphical description of a finite state machine into a behavioral description in either ABEL or VHDL. In this section, the STATMACH schematic macro in the STACK schematic is replaced with a state machine macro.

Creating a State Machine Macro

1. Click on the State Editor button in the Project Flowchart. A dialog box appears.

2. Click OK to use the Design Wizard.

3. Click Next.
4. Make sure that VHDL is selected, then click Next.
5. Specify a file name of STATEMAC, then click Next.
6. Add the following input and output ports. This procedure is identical to that used to create the ANDBLK2 symbol. When all ports have been added, click Next.
   - Inputs: STACKEN, PUSH, FULL, EMPTY, CLK, RESET
   - Outputs: UPDOWN, WRITEN, ADDREN
7. Make sure that One state machine is selected, then click Finish.

The Design Wizard defines the inputs and outputs of the state machine, based on the information entered in step 6. It also determines that CLK is the clock input, based on the port name. The default name of the state register is Sreg0.

Defining States

1. Select **FSM** → **State** or click on the State button in the vertical toolbar.

Place the state bubble as shown below. The default state name is S1.

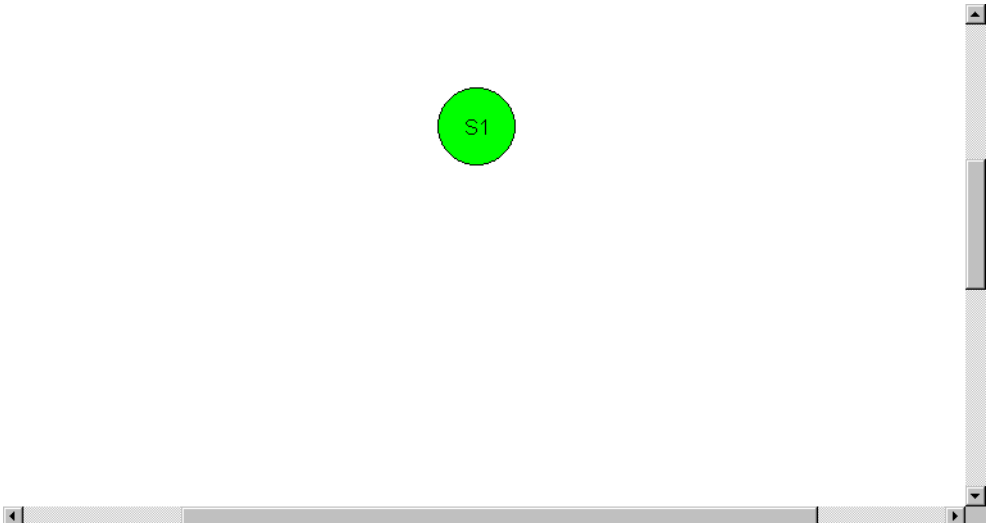


Figure 4-32 Placing a State

2. Click on the state name to select it, then click again to edit the text.
3. Type WAITPUSH and press Return. Now the state name extends outside the state bubble.
4. Click on the state bubble to select it. Click and drag the small squares to change the size and shape of the bubble. When the state bubble is large enough, click and drag the state name to center it in the bubble.
5. Repeat steps 1-4 to create a second state named HANDLEPUSH.

   These are the only states needed for this state machine.

   To ensure that the state machine powers up in the correct state, you must define an asynchronous reset condition. This reset will not be connected in the schematic, but its presence directs the VHDL compiler to define the state encoding so that the machine will power up in the correct state.

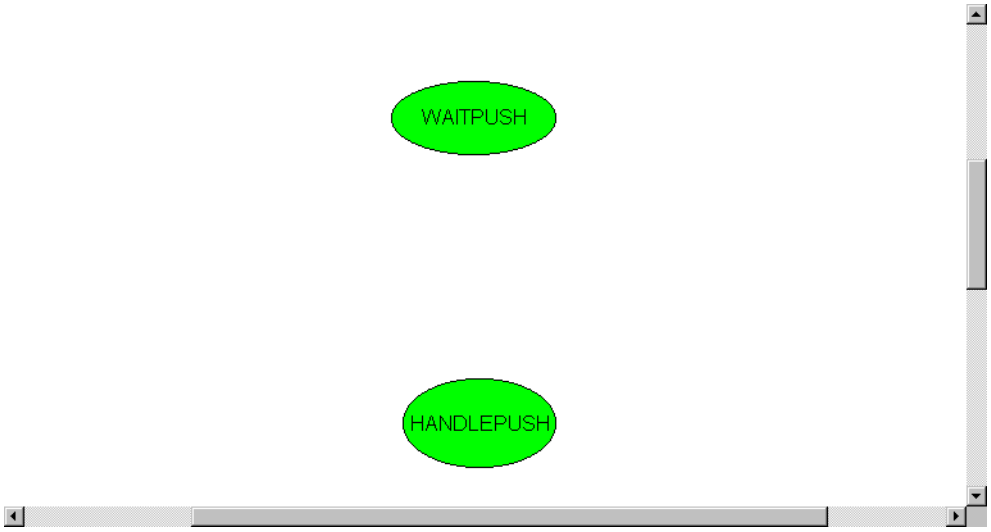


Figure 4-33 WAITPUSH and HANDLEPUSH States

6. Select **FSM** → **Reset** or click on the Reset button in the vertical toolbar.

7. Place the reset symbol as shown in the figure. Click inside the WAITPUSH state bubble to define this as the reset state.
8. To define the reset as asynchronous, right-click on the reset symbol and select Asynchronous.

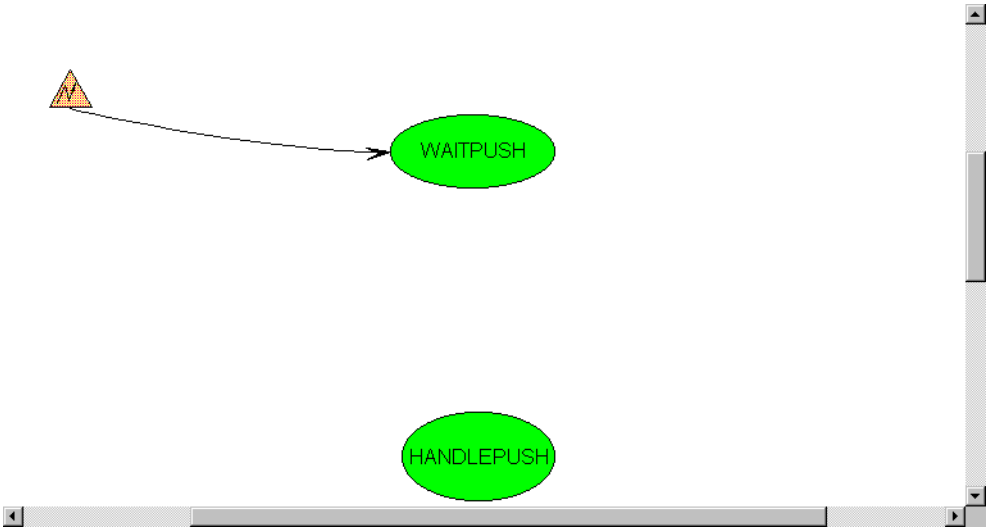


Figure 4-34 State Machine with Reset Defined

Defining Transitions, Conditions and Actions

Now it is time to define transitions, conditions, and actions.

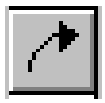
Transitions define the movement from one state to another. They are drawn as arrows between state bubbles.

If there is more than one transition leaving a state, you must associate a condition with each transition. A condition is a Boolean expression. When the condition is true, the machine moves along the transition arrow.

Actions are HDL statements that are used to make assignments to output ports or internal signals. Actions can be executed at several points in the 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.

Note: Conditions and actions must be defined using the syntax of the target HDL (ABEL or VHDL).

1. Select **FSM** → **Transition** or click on the Transition button in the vertical toolbar.



2. Click inside the WAITPUSH state bubble to start the transition. To create a bend in the arrow, move down and to the left and click. Click inside the HANDLEPUSH state bubble to complete the transition.

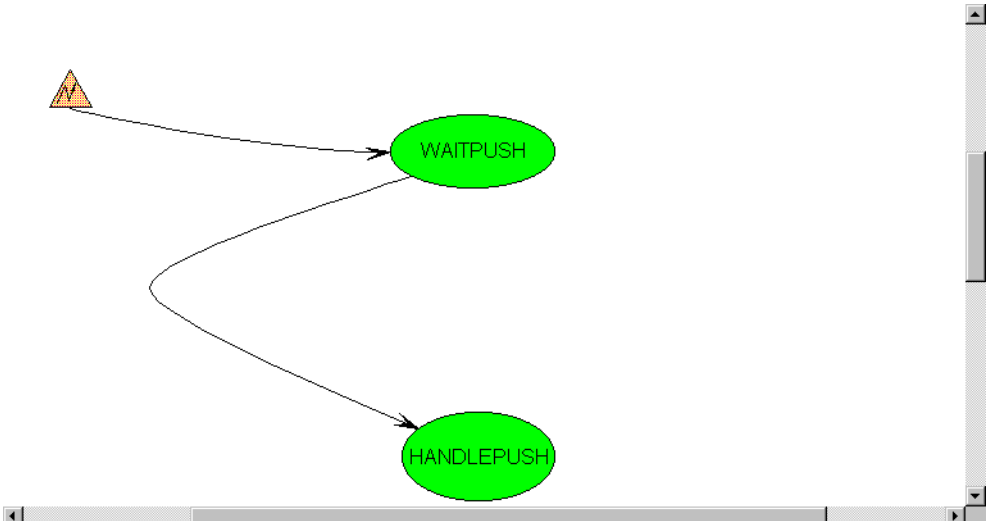
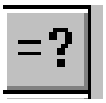


Figure 4-35 Drawing a Transition

3. Select **FSM** → **Condition** or click on the Condition button in the vertical toolbar.



4. Click on the transition arrow. A text box appears.
5. Enter the following condition.

   ```
   STACKEN='1' and PUSH='1' and FULL='0'
   ```

6. Select **FSM** → **Action** → **Transition** or click on the Transition Action button in the vertical toolbar.

7. Click on the transition arrow. A text box appears.
8. Enter the following actions. Click anywhere outside the text box to end the entry.

```
ADDREN <= '1';
UPDOWN <= '1';
WRITEN <= '1';
```

9. Click and drag the condition and action text boxes to the positions shown below.

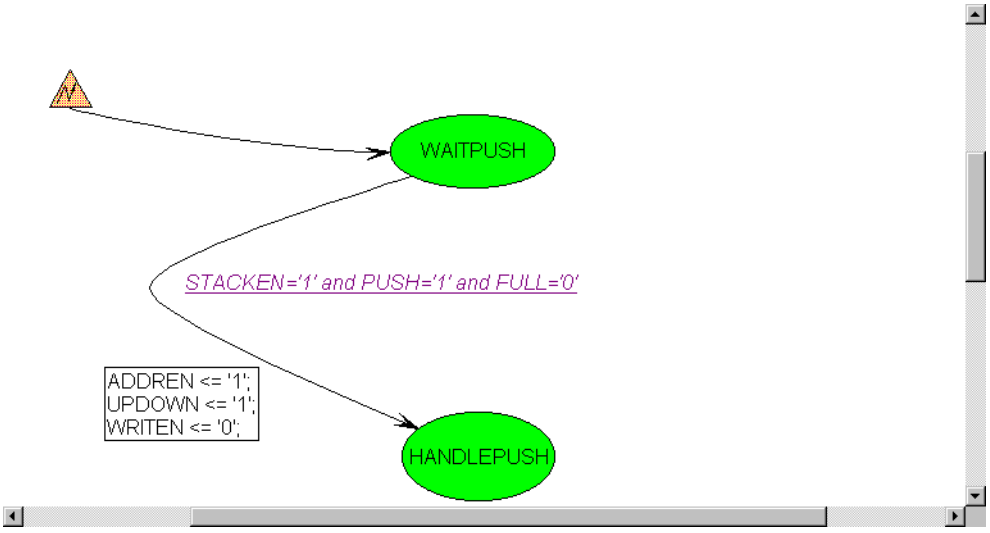


Figure 4-36 Adding Conditions and Actions

10. Add the remaining transitions, conditions, and actions as shown below.

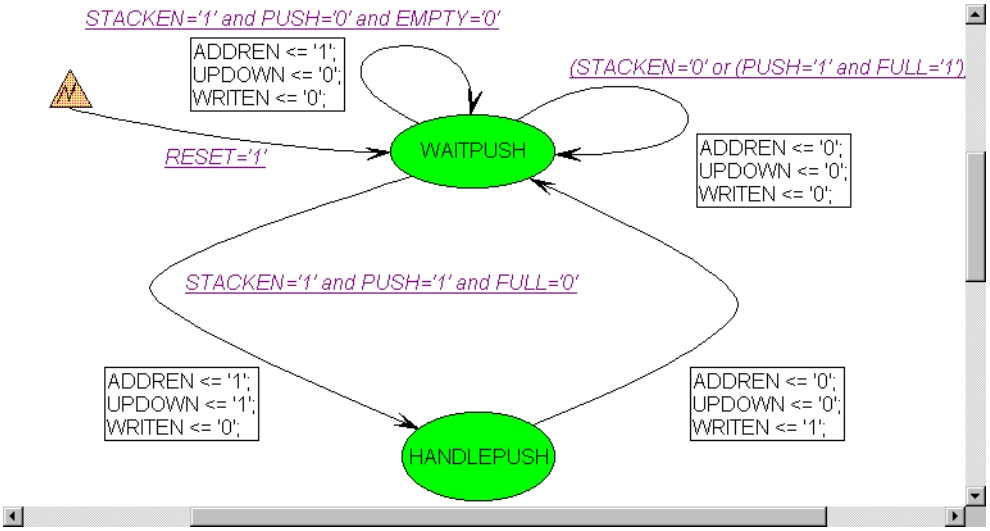


Figure 4-37 Completed State Diagram

Note: The transition from the HANDLEPUSH state to the WAITPUSH state does not require a condition, since it is the only transition leaving that state.

11. Select **File** → **Save** to save the state diagram.

Generating and Compiling VHDL Code

1. Select **Project** → **Create Macro**.

   The state diagram is converted into VHDL code, which is compiled, and a symbol is generated. A dialog box will pop up to inform you that the symbol has been successfully created.

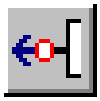
2. Click OK to close the dialog box.

3. Select **File** → **Exit** to close the State Editor.

Placing the Macro

The final step is to replace the schematic-based component with the state machine macro.

1. Open the Calc schematic and push into the STACK symbol.
2. Select the STATMACH component and click on the Disconnect Symbol button in the horizontal toolbar.



3. Press Del to delete the component.
4. Place the STATEMAC symbol in the space left by the STATMACH. The pins may not line up with the hanging nets.
5. Drag the hanging nets to connect them to the appropriate symbol pins.

Using the HDL Editor and XVHDL (Optional)

Note: XVHDL is only supported in Foundation Base-VHDL, Standard-VHDL, and Foundation Express packages.

The HDL Editor facilitates the creation of text-based macros by color-coding the source file, providing a syntax checker, and supplying language templates for common constructs. In this section, the SEG7DEC schematic macro is replaced with a VHDL macro.

Creating a VHDL Macro

1. Click on the HDL Entry button in the Project Flowchart. A dialog box appears.

2. Click OK to use the Design Wizard.
3. Click Next.
4. Make sure that VHDL is selected, then click Next.

5. Specify a file name of SEG7DECV, then click Next.
6. Add the following input and output ports. This procedure is identical to that used to create the ANDBLK2 symbol. When all ports have been added, click Finish.
   - Inputs: Q[3:0]
   - Outputs: A, B, C, D, E, F, G

The Design Wizard creates a “skeleton” VHDL file, based on the information entered in step 6. VHDL keywords are displayed in red, and comments are displayed in green. The “skeleton” SEG7DECV VHDL code created by the Design Wizard is shown below:

```
library IEEE;
use IEEE.std_logic_1164.all;

entity seg7decv is
   port (
      q: in STD_LOGIC_VECTOR (3 downto 0);
      a: out STD_LOGIC;
      b: out STD_LOGIC;
      c: out STD_LOGIC;
      d: out STD_LOGIC;
      e: out STD_LOGIC;
      f: out STD_LOGIC;
      g: out STD_LOGIC
   );
end seg7decv;

architecture seg7decv_arch of seg7decv is
begin

   -- <<enter your statements here>>

end SEG7DECV_arch;
```

Creating and Editing VHDL Code

1. Place the cursor at the beginning of the line reading:

   ```
   -- <<enter your statements here>>
   ```

2. From the menus, select **Tools** → **Language Assistant**. The Language Assistant window appears. The Language Assistant contains two types of templates.

   - Language Templates—samples of common VHDL constructs, such as IF statements.
   - Synthesis Templates—samples of commonly used logic functions, such as counters.

3. Click on the "+" next to Synthesis templates to view the list of available templates.

4. Select the HEX2LED Converter. The template is displayed in the right half of the window.

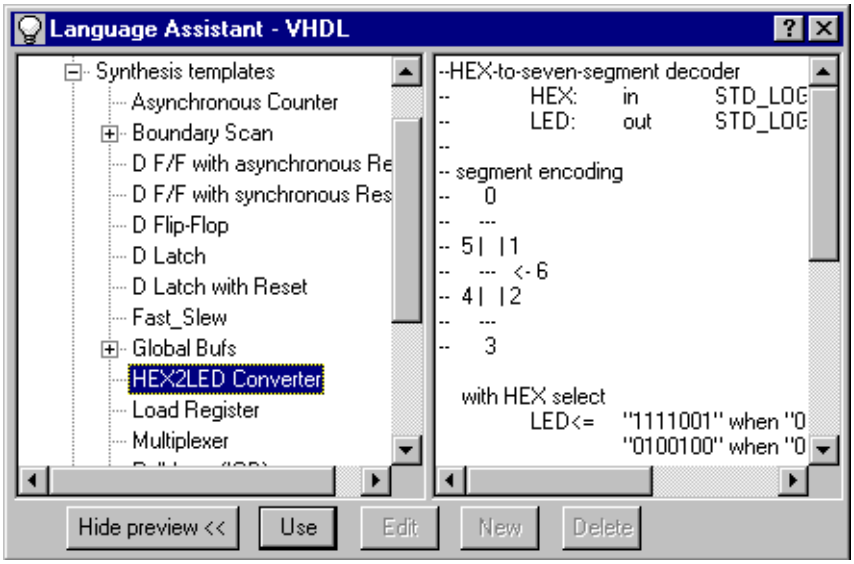


Figure 4-38 Language Assistant Window

5. Click Use to insert the template into your VHDL code.

The template takes a 4-bit vector called HEX and outputs a 7-bit vector called LED. These names do not match the port names used for the SEG7DECV macro.

6. Make the necessary edits to complete the SEG7DECV macro. The completed VHDL source code is shown below:

```
library IEEE;
use IEEE.std_logic_1164.all;

entity seg7decv is
   port (
      q: in STD_LOGIC_VECTOR (3 downto 0);
      a: out STD_LOGIC;
      b: out STD_LOGIC;
      c: out STD_LOGIC;
      d: out STD_LOGIC;
      e: out STD_LOGIC;
      f: out STD_LOGIC;
      g: out STD_LOGIC
   );
end seg7decv;

architecture seg7decv_arch of seg7decv is
signal led: std_logic_vector (6 downto 0);

begin

--HEX-to-seven-segment decoder
--      HEX: in STD_LOGIC_VECTOR (3 downto 0);
--      LED: out STD_LOGIC_VECTOR (6 downto 0);
--
-- segment encoding
--       0
--      ---
--  5 |   | 1
--      ---   <- 6
--  4 |   | 2
--      ---
--       3

   with q select
      LED<= “1111001” when “0001”,  --1
            “0100100” when “0010”,  --2
            “0110000” when “0011”,  --3
            “0011001” when “0100”,  --4
            “0010010” when “0101”,  --5
            “0000010” when “0110”,  --6
```

```
                    "1111000" when "0111",  --7
                    "0000000" when "1000",  --8
                    "0010000" when "1001",  --9
                    "0001000" when "1010",  --A
                    "0000011" when "1011",  --B
                    "1000110" when "1100",  --C
                    "0100001" when "1101",  --D
                    "0000110" when "1110",  --E
                    "0001110" when "1111",  --F
                    "1000000" when others;  --0

-- Assign LED[6:0] to outputs A through G

   A <= LED(0);
   B <= LED(1);
   C <= LED(2);
   D <= LED(3);
   E <= LED(4);
   F <= LED(5);
   G <= LED(6);

end seg7decv_arch;
```

7. Select **File** → **Save** to save the file.

Compiling with XVHDL

1. Select **Project** → **Create Macro**.

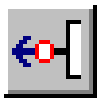
   The VHDL code is compiled, and a symbol is generated. A dialog box will pop up to inform you that the symbol has been successfully created.

2. Click OK to close the dialog box.
3. Select **File** → **Exit** to close the HDL Editor.

Placing the Macro

The final step is to replace the schematic-based component with the VHDL macro.

1. Open the Calc schematic and select the SEG7DEC component.
2. Click on the Disconnect Symbol button in the horizontal toolbar.



3. Press Del to delete the component.
4. Place the SEG7DECV symbol in the space left by the SEG7DEC. The pins may not line up with the hanging nets.
5. Drag the hanging nets to connect them to the appropriate symbol pins.

Other Special Components (Optional)

This section describes some optional components.

The STARTUP Block (XC4000 Family Only)

The STARTUP block allows some aspects of the design to be controlled globally. In this section, STARTUP is used to connect an external signal to the global set/reset net which is built into the XC4000E architecture. This global net connects to all flip-flops in the device and sets or resets them asynchronously (set or reset is determined at the flip-flop level). An advantage to using the global net is that no routing resources are used. For more information on STARTUP, see the online *Libraries Guide*.

The STARTUP symbol is used here to implement a system-wide reset signal called NOTGBLRESET. This signal is active-Low; therefore, when NOTGBLRESET is low, the Calc circuitry is reset.

1. In the Calc schematic, add the components, nets, and labels as shown in the figure. An inverter is added to the signal path because the GSR pin on STARTUP is active-High. Also, since GSR is *implicitly* connected to all reset logic throughout the device, GBLRESET is connected only to the GSR pin on the STARTUP symbol and is not explicitly connected to any flip-flops in the design.

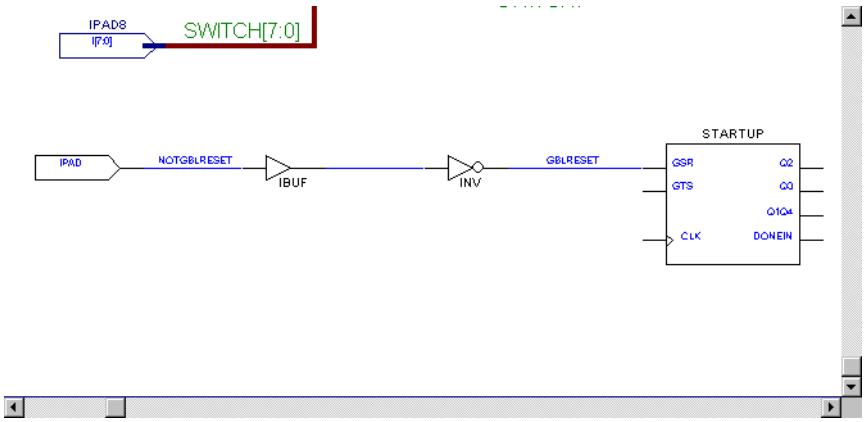


Figure 4-39 Adding the STARTUP Symbol

2. Save the Calc schematic.

The CONFIG Symbol

The CONFIG symbol can be used to document implementation options on the schematic. In this section, the CONFIG symbol will be used to specify a fast configuration rate (FAST is an option to the BitGen program).

1. Place the CONFIG symbol in the lower right corner of the Calc schematic.
2. Double click on the CONFIG symbol to add parameters.
3. Under the Parameters section, enter the Name as “ConfigRate” and the Description as “FAST” and click ADD.
4. Double click on the parameter until two dots are shown, then click Apply.
5. Click MOVE to adjust the placement of the parameter on the schematic. Place the parameter text within the CONFIG symbol as shown below.

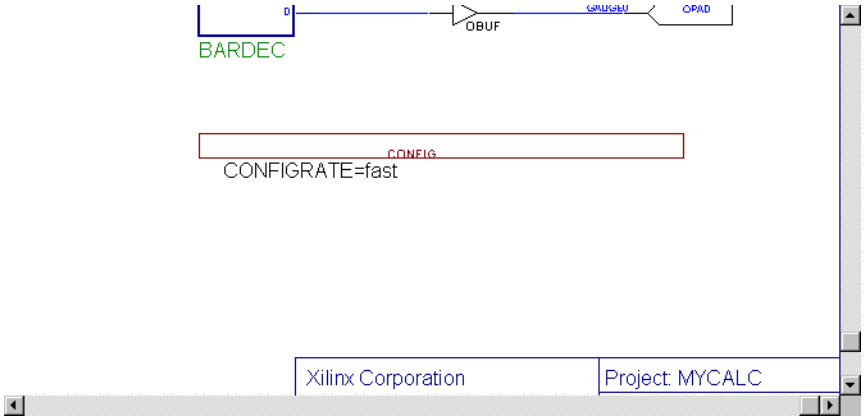


Figure 4-40 Adding the CONFIG Symbol

6. Save the Calc schematic.

Using a Constraints File

A constraints file supplies constraints information in a textual form. Sometimes this method is more efficient than putting constraints on a schematic. An sample constraints file is shown below; this is the user constraints file, calc.ucf, which is supplied with this tutorial. The constraints file syntax is the same for all device families.

The place-and-route software must be instructed to read and apply the .ucf file when the design is read into the Design Manager. This procedure is detailed in the "Using Constraint Files" section of the "Foundation Overview" chapter.

```
inst SWITCH<7>     LOC=p19;
inst SWITCH<6>     LOC=p20;
inst SWITCH<5>     LOC=p23;
inst SWITCH<4>     LOC=p24;
inst SWITCH<3>     LOC=p25;
inst SWITCH<2>     LOC=p26;
inst SWITCH<1>     LOC=p27;
inst SWITCH<0>     LOC=p28;

inst A             LOC=p49;
inst B             LOC=p48;
inst C             LOC=p47;
inst D             LOC=p46;
inst E             LOC=p45;
inst F             LOC=p50;
inst G             LOC=p51;
inst OFL           LOC=p41;

inst GAUGE<3>      LOC=p61;
inst GAUGE<2>      LOC=p62;
inst GAUGE<1>      LOC=p65;
inst GAUGE<0>      LOC=p66;

inst STACKLED<3>   LOC=p57;
inst STACKLED<2>   LOC=p58;
inst STACKLED<1>   LOC=p59;
# inst STACKLED<0> LOC=p60;

inst NOTGBLRESET   LOC=p56;
```

Functional Simulation

Functional simulation is performed before design implementation to verify that the logic that you have created is correct.

Starting the Logic Simulator

1. Click on the SIM Funct button in the Project Flowchart. You will be prompted to update the schematic netlist, because you modified the schematic but did not write out a netlist.

2. Click Yes. A functional simulation netlist is created and loaded into the Logic Simulator.

The Logic Simulator opens a Waveform Viewer window with Simulator toolbar.

Waveform Viewer

All simulation information is displayed in this window:

- signals that are being monitored
- stimulators applied to signals
- signal states at the selected point in time
- waveforms of signals over time

The Waveform Viewer also contains a toolbar.

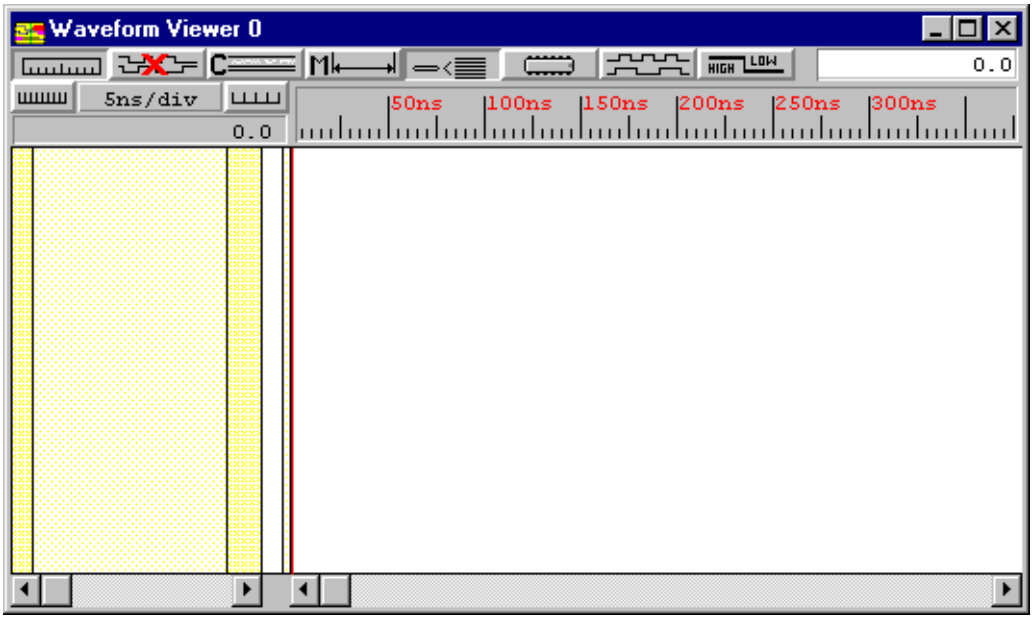


Figure 4-41 Waveform Viewer

Simulator Toolbar

This toolbar contains buttons for common simulation operations, such as running or restarting the simulation.

Figure 4-42 Simulator Toolbar

Selecting Nets to Probe

You can select the signals that you would like to monitor during simulation either through the Logic Simulator interface or from the Schematic Editor.

Adding Probes From the Logic Simulator

1. Select **`Signal`** → **`Add Signals`** or click on the Select Component button in the Waveform Viewer. The Component Selection window appears.

This window is divided into three panes. The leftmost pane lists buses and nets on the selected level of hierarchy. The center pane lists the schematic components on the selected level of hierarchy. The rightmost pane displays the design hierarchy. The top level (Root) is selected by default.

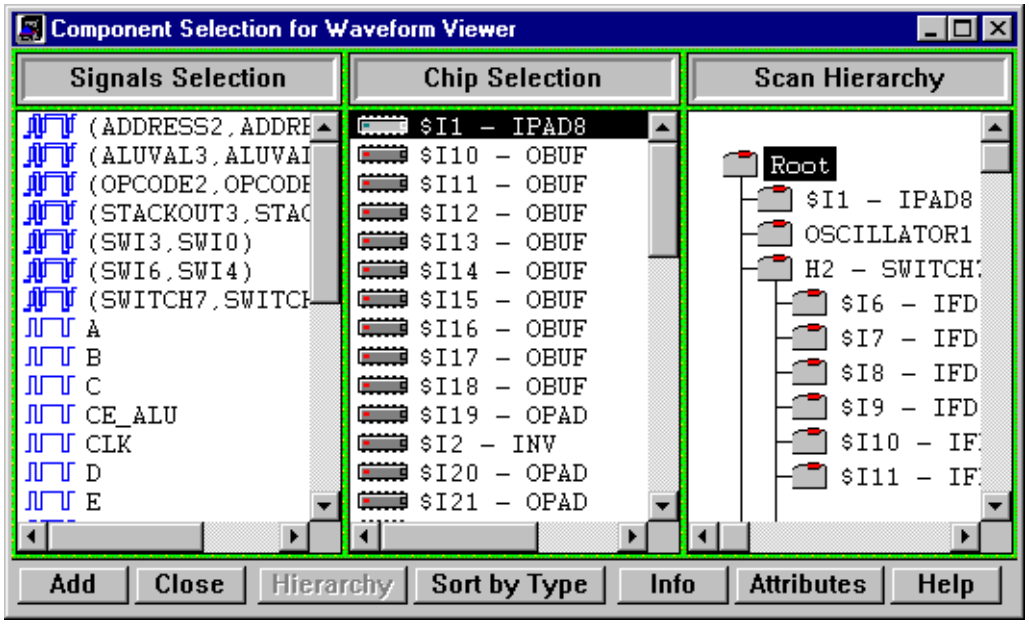
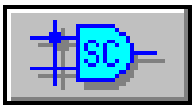


Figure 4-43 Component Selection Window

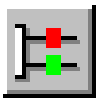
2. In the leftmost pane, double click on CLK. A red check mark appears next to the signal name to indicate that the signal is being displayed in the Waveform Viewer.
3. Double click on the following buses and nets to add them to the Waveform Viewer:
   - (ALUVAL3, ALUVAL0)
   - (STACKOUT3, STACKOUT0)
   - (SWITCH7, SWITCH0)
   - CLK
   - PUSH
   - STACKEN
4. Click Close to close the Component Selection window.

Adding Probes From the Schematic Editor

1. To switch to the Schematic Editor, select **`Tools`** → **`Schematic Capture`** or click on the SC button in the horizontal toolbar The Schematic Editor becomes the active window. If the Schematic Editor is not running, it is opened and calc.sch is loaded.



2. Select **`Mode`** → **`Testpoints`**. This brings up the SC Probes toolbox. The Probe tool is selected by default.



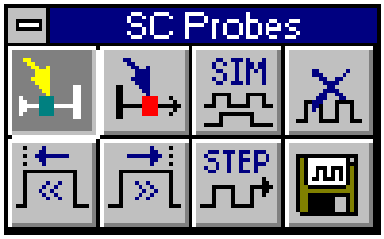


Figure 4-44 SC Probes Toolbox

3. Find the EXEC net (output of the DEBOUNCE component). Click on the net label to add a probe to the net.

   The probe appears as a small gray box next to the net label. There are also probes on the buses and nets that were selected from the Logic Simulator. When the simulation is running, the gray boxes will change color to reflect the current state of the net. Probes on buses do not change color; the bus value is displayed (in hexadecimal) inside the gray box.

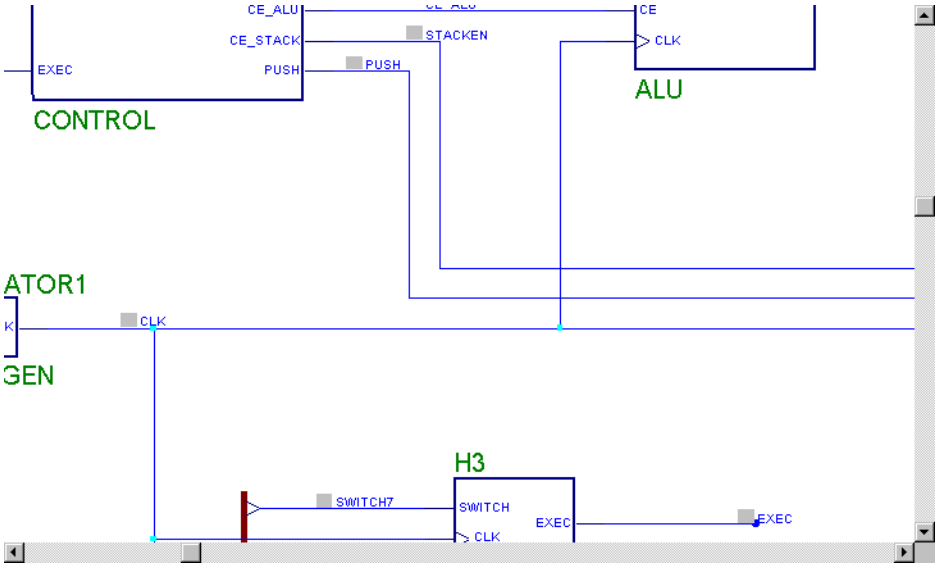


Figure 4-45 Calc Schematic With Simulation Probes

4. To switch back to the Logic Simulator, select **File** → **Go To Simulator** or click on the Simulator button in the horizontal toolbar or the SC Probes toolbox. The probe you placed from the Schematic Editor is now listed in the Waveform Viewer.

Manipulating Buses

The SWITCH[7:0] bus is more conveniently viewed as SWITCH[6:0] (the opcode bits) and SWITCH7 (the execute switch).

1. Select the SWITCH7 bus in the Waveform Viewer.
2. Right-click to bring up the Signal menu and select **Bus** → **Flatten**. This breaks the bus up into individual signals.
3. Select the SWITCH6 signal.
4. Hold down the Shift key and select the SWITCH0 signal to select all seven signals.
5. Right-click and select **Bus** → **Combine**.

   You can see what signals comprise a bus without flattening it.
6. Select **View** → **Buses** or click on the Bus On / Off button in the Waveform Viewer.

   All buses in the Waveform Viewer are expanded to show the individual signals. The LSB of a bus is denoted by a "*" and the MSB is denoted by a "$". This determines how the bus value is displayed in the Waveform Viewer and how stimulus values are applied to the bus (if it is an input). All other signals that are part of a bus are denoted by a "+".

   Notice that the SWITCH6 bus is currently backwards (bit 6 is the LSB and bit 0 is the MSB).
7. Click on the Bus On / Off button again to collapse the buses into a single line in the Waveform Viewer.

8. Select the SWITCH6 bus.
9. Right-click and select **Bus** → **Change Direction**.
10. Expand the buses again to verify that SWITCH6 is now the MSB. Collapse the buses when you are done.

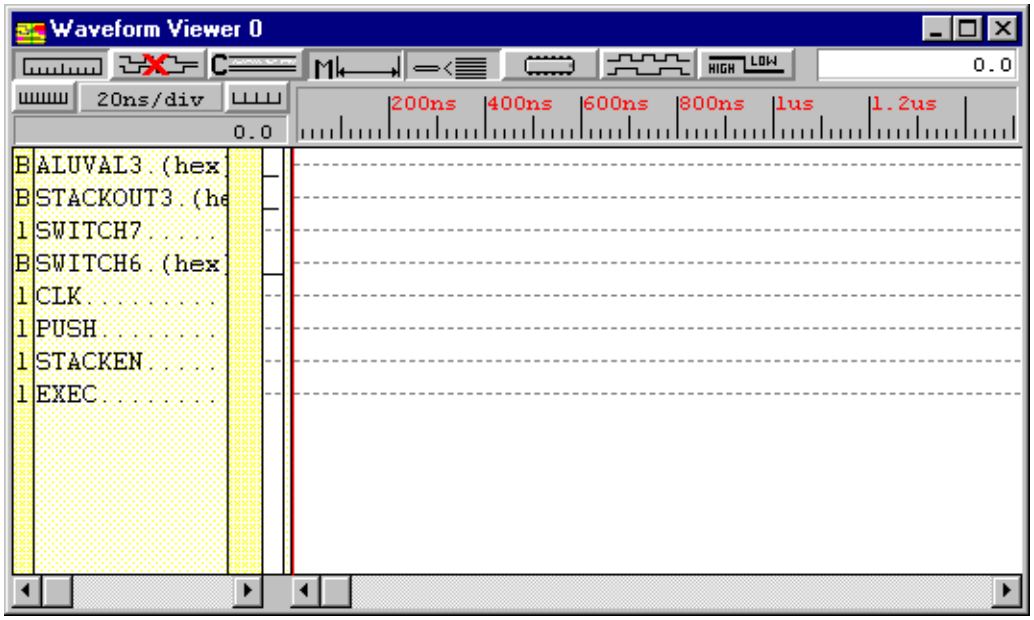
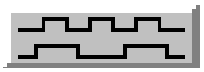


Figure 4-46 Selected Nets and Buses

Assigning Stimulators

The Logic Simulator provides some basic stimulators: simple clocks, keyboard (interactive) stimulators, and constants. To define complex waveforms, you can create your own stimulators.

Select **Signal** → **Add Stimulators** or click on the Select Stimulus button in the Waveform Viewer. The Stimulator Selection window appears.



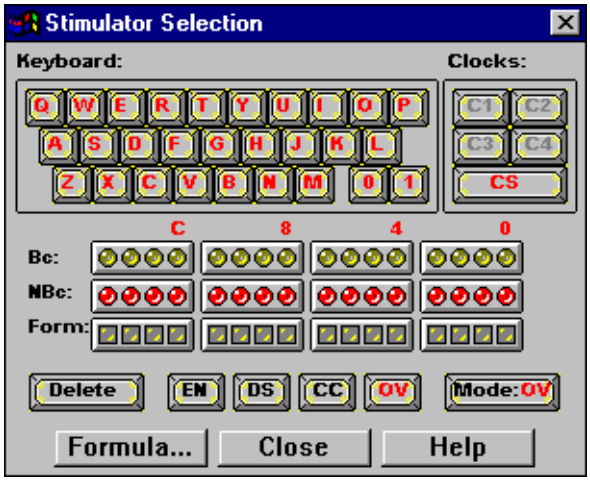


Figure 4-47 Stimulator Selection Window

This window contains different types of stimulators.

- The Keyboard section contains the interactive stimulators. Pressing the appropriate keyboard letter will toggle the stimulator between 0 and 1. The keyboard area also contains the stimulators for a constant 0 or 1.
- The Clocks section is used to assign user-defined clock waveforms. This section also contains the Custom Select (CS) button. Refer to the Logic Simulator online help for more information about these buttons.
- The two rows of round LEDs represent the bits of a free-running 16-bit counter (true and inverted bits). This counter can be used to generate clock signals of various frequencies with a 50% duty cycle.
- The row of square LEDs is used to assign user-defined formulas. Up to 16 formulas can be assigned to these LEDs.

The buttons along the bottom of the window are used to delete, enable, disable, or change the drive strength of stimulators. Refer to the Logic Simulator online help for more information about these buttons.

Defining the Clock

1. In the Waveform Viewer, select the CLK signal.
2. In the Stimulator Selection window, click on the far right yellow LED (B0). The Waveform Viewer updates to show that CLK will be driven by the B0 stimulator.

   The frequency of the free-running counter can be customized to suit your needs. For this design, you want CLK to have a 100 ns period. To force the simulation to run faster, you can also change the simulation precision to 100 ps.

Note: For functional simulation, you could change the precision to 1 ns or more. For timing simulation, Xilinx recommends keeping the precision at either 10 ps or 100 ps to ensure that all delays are simulated accurately.

3. From the menus, select **`Options`** → **`Preferences`**. The Preferences dialog box appears.
4. Click on the Simulation Precision drop-down list and select 100ps.
5. Click on the B0 Period drop-down list and select 100ns.
6. Click OK.

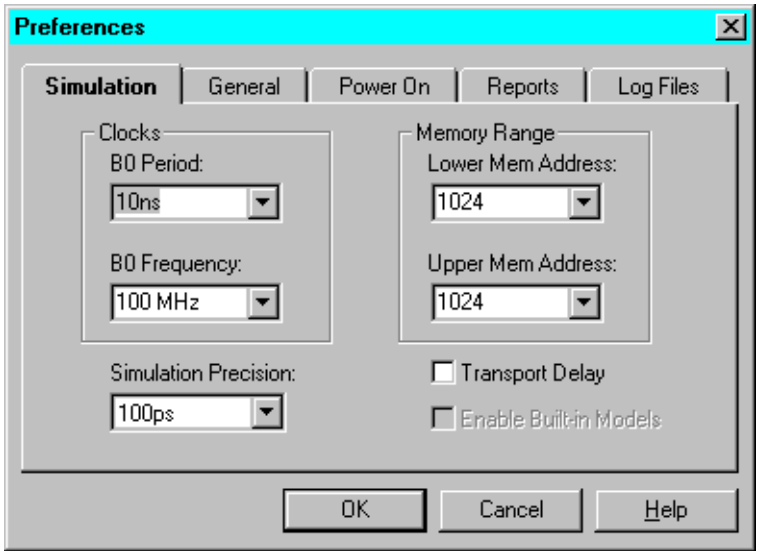


Figure 4-48 Setting Simulation Preferences

Defining Formulas

1. In the Stimulator Selection window, click on Formula. The Set Formula dialog appears.
2. Double click on F0 to select it.
3. In the Edit Formula field, type the following:

   `[0]200[61]500[0d]500[7b]600[3f]500[7b]500[50]500`

4. Click on Accept. The formula is displayed in the Formula Stimulators list.

   This defines F0 as a bus stimulator which has value 0 for 200 ns, 61 (hex) for 500 ns, etc. This formula will be used to stimulate the SWITCH[6:0] set of inputs to perform the following commands:

   00: ADD 0 to register value (the ALUVAL bus should be 0).
   61: LOAD register with the value 1 (ALUVAL = 1).
   0D: ADD D (13) to register value (ALUVAL = E).
   7B: PUSH register value to stack (STACKOUT = E).
   3F: XOR register value with F (ALUVAL = 1).
   7B: PUSH register value to stack (STACKOUT = 1).
   50: CLEAR register value (ALUVAL = 0).

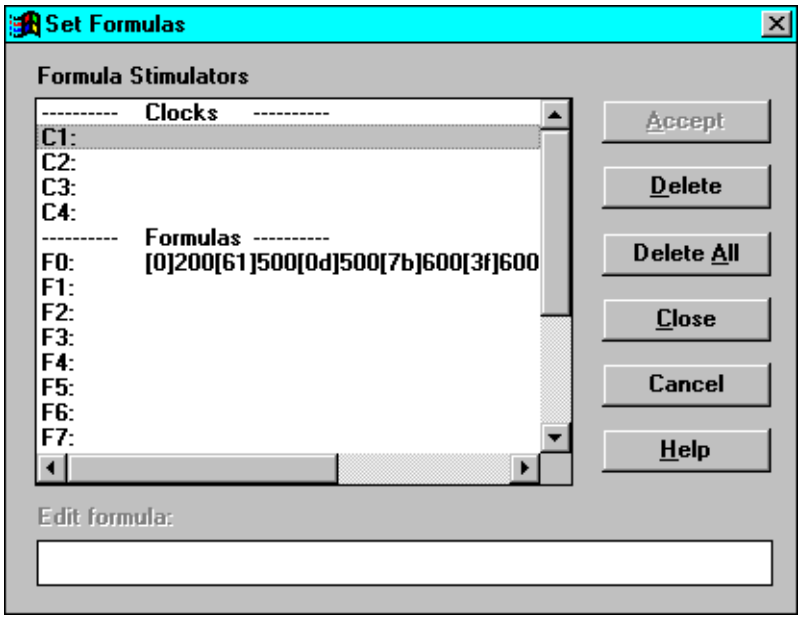


Figure 4-49 Defining a Formula

5. Double click on F1.
6. In the Edit Formula field, type the following and click on Accept:

   ```
   H200 (L200H300)2 L200H400 (L200H300)3
   ```

   This defines F1 as a stimulator which is high for 200 ns, then repeats the following pattern twice: low for 200 ns, then high for 300 ns. Then it is low for 200 ns, etc.

Note: Spaces were used to make the formula syntax easier to understand. Spaces are ignored by the simulator and will be removed from the formula when you click Accept.

7. Click on Close.

8. In the Waveform Viewer, select the SWITCH6 bus.
9. In the Stimulator Selection window, click on the far right square LED (F0).
10. Repeat steps 8-9 to assign the F1 stimulator to SWITCH7.
11. Click on Close.

Saving the Input Waveforms

Now that you have selected the nets and buses you wish to view, and have assigned stimulators, you can save this information as a test vector (.TVE) file. You can load the test vectors for timing simulation, or for functional simulation after making design changes (as long none of the selected nets have been deleted or renamed).

1. Select **File** → **Save Waveform**.
2. In the File Name field, type “calcsim1.tve” and click OK.

Simulating the Circuit

The Simulator toolbar contains two buttons for running the simulation: Short and Long. These buttons can be customized to run the simulation for any length of simulation time.

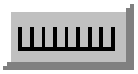
Figure 4-50 Short Step

Figure 4-51 Long Step

1. Select **Options** → **Simulation Step**. The Step dialog box appears.
2. Use the drop-down lists to set the Short Step to 200 ns and the Long Step to 1 us (1000 ns).

Figure 4-52 Customizing the Simulation Step Buttons

3. Click Set Step to confirm the settings, then click Close.
4. To run the simulation, click the Long button three times and the Step button once.
5. To get a better view of the waveforms, click on the Zoom Out button in the Waveform Viewer until the time scale reads `50ns/div`.



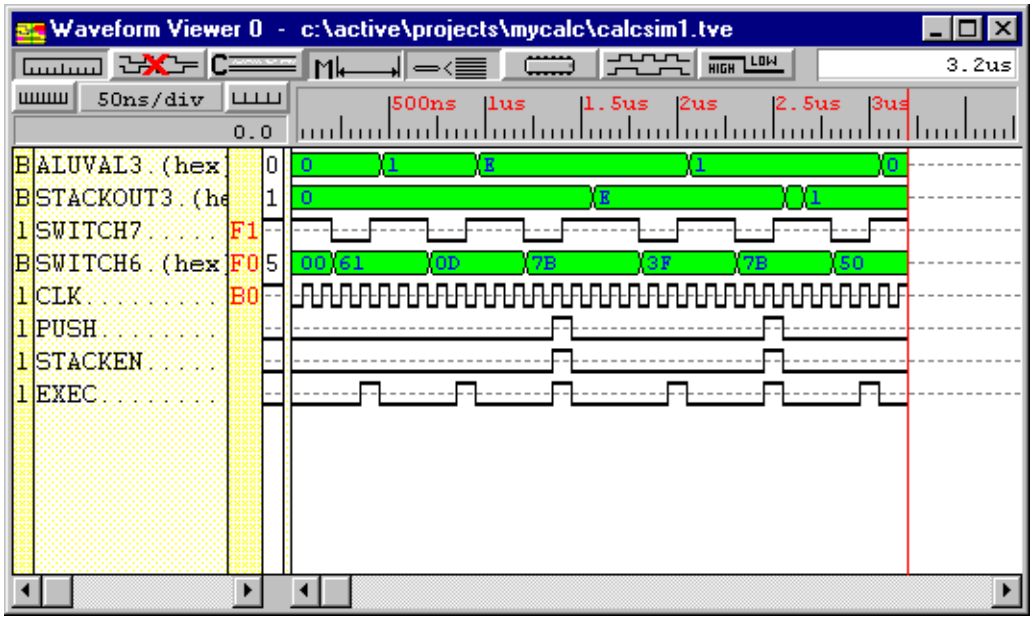


Figure 4-53 Simulation Results

6. Switch back to the Schematic Editor to see the probe values annotated directly on the schematic.

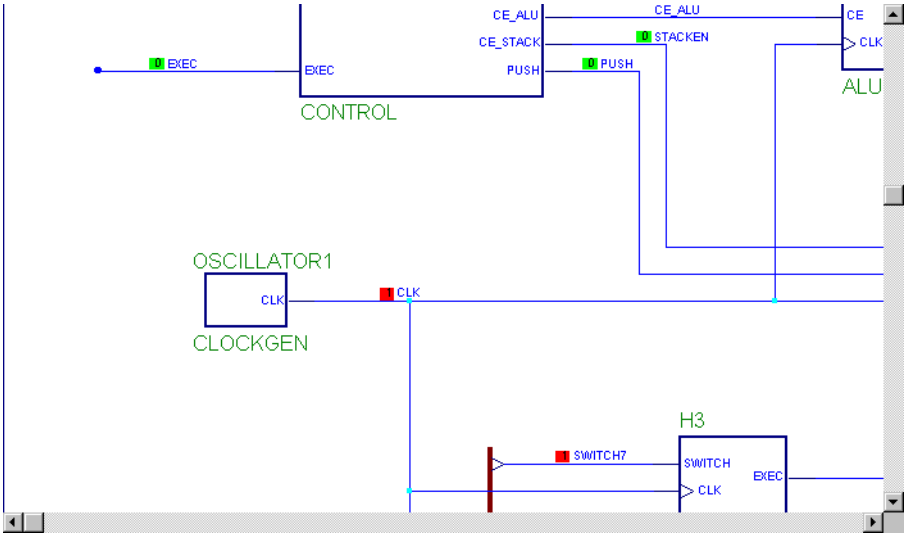


Figure 4-54 Probe Values on the Schematic

7. Select **File** → **Exit** to close the Schematic Editor.
8. In the Logic Simulator, select **File** → **Exit**.

Using the Design Implementation Tools

The design implementation tools take a design, represented by an EDIF file, and implement it in an FPGA or CPLD. You can also use the design implementation tools to generate timing information that you can import into the Logic Simulator.

This section gives a brief overview of the design implementation flow. For a more in-depth discussion of the flow, including advanced implementation options, see the online *Development System Reference Guide*.

1. Click on the Implement M1 button in the Project Flowchart to invoke the Design Implementation Tools. The Project Manager performs the following steps:
   - Creates an EDIF netlist representing the design (this step is not performed if a current EDIF netlist exists).
   - Starts the Xilinx Design Manager (if the Design Manager is already running, it becomes the active application).

The Design Manager is a graphical design-flow and revision manager. Each project has associated with it objects known as “versions” and “revisions.” Versions represent logic changes in a design (for example, adding a new block of logic, replacing an AND gate with an OR gate, or adding a flip-flop); revisions represent different executions of the design flow on a single design version, usually with new implementation options (for example, higher place-and-route effort, a change in part type, or experimentation with new bitstream options). In the next step, you make a new version and revision on which you run the implementation design flow.

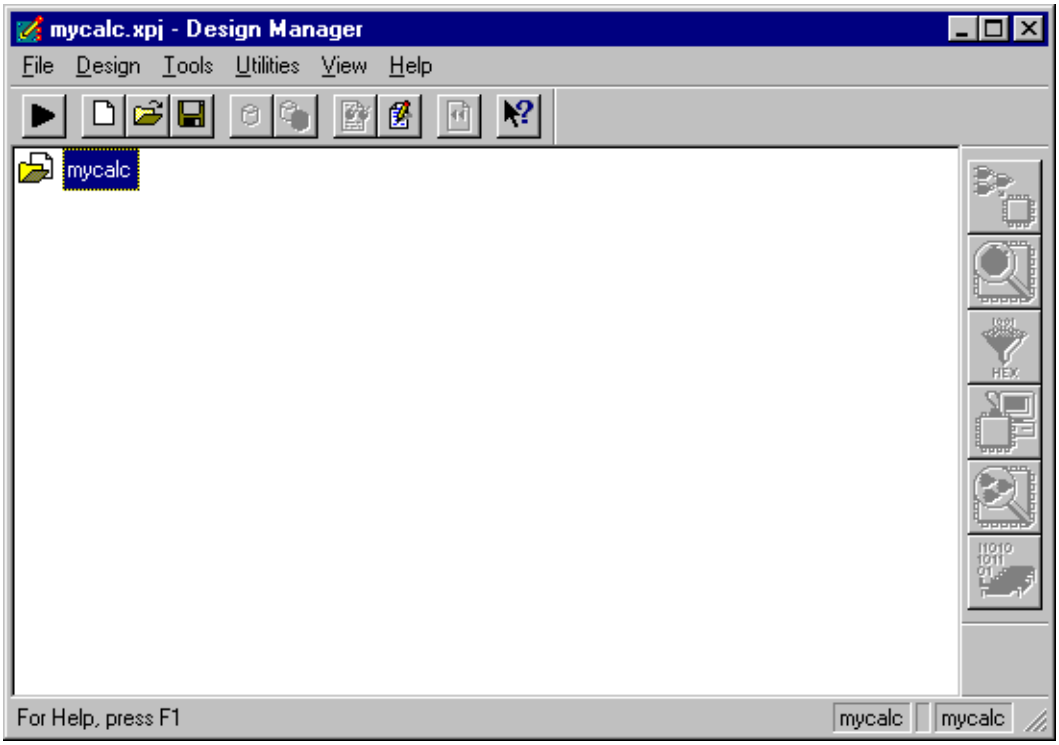


Figure 4-55 Design Manager

2. Within the Design Manager, select **`Design`** → **`Implement`**. This brings up the Implement dialog box.

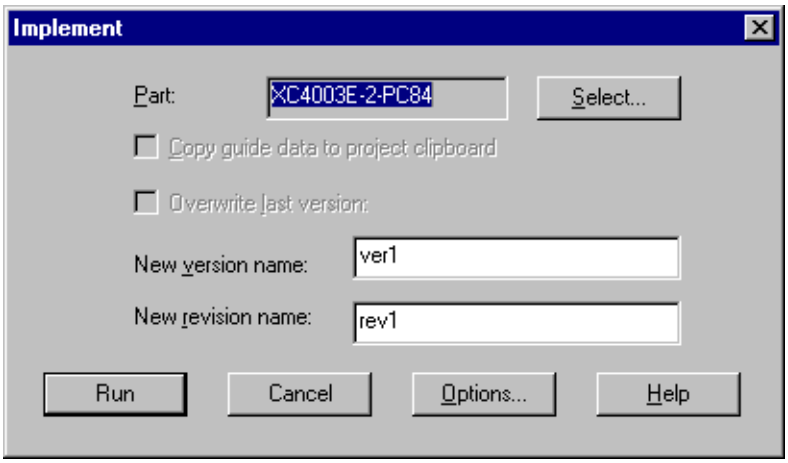


Figure 4-56 Implement Dialog Box

3. The Project Manager writes the target part into the EDIF netlist. If you wish to target a different device, click Select to display a listing of available devices. Select a family, device, package, and speed grade, then click OK. The part number is inserted into the Part field in the Implement dialog box.
4. Click on Options. The Options dialog box appears.

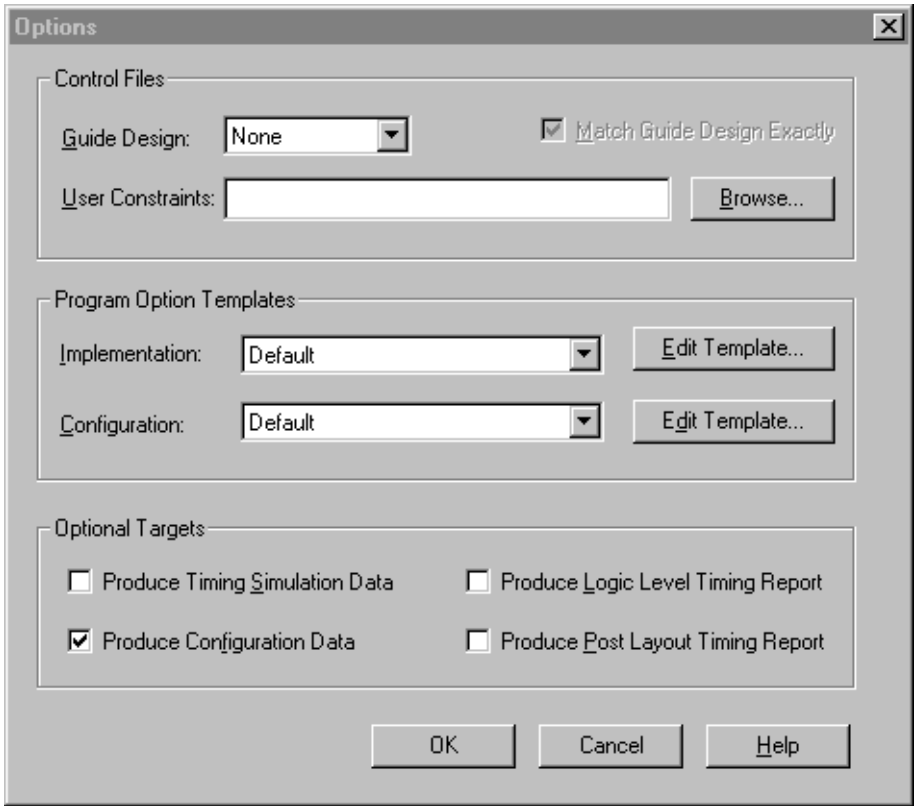


Figure 4-57 Options Dialog Box

5. Click Browse by the User Constraints field. Select the calc.ucf file from the project directory, then click OK.
6. Under Optional Targets, make sure the following are selected:
   - Produce Timing Simulation Data: This generates a back-annotated EDIF netlist for timing simulation.
   - Produce Configuration Data: This generates a programming bitstream suitable for downloading into the Xilinx device.
   - Produce Post Layout Timing Report: This generates a timing report file based on how the design is actually routed.

You can also select the following option:

Produce Logic Level Timing Report: This generates a preliminary (pre-place-and-route) timing report based on the number of logic levels in each signal path. Since it is generated before the place-and-route layout step, it only contains estimated delays for device routing. Looking at this report before place-and-route can be useful for determining if your timing requirements can be met.

7. Click OK to return to the Implement dialog box.
8. Verify that the version is “ver1” and the revision is “rev1”. Then click Run. The Flow Engine comes up as shown in the figure below.

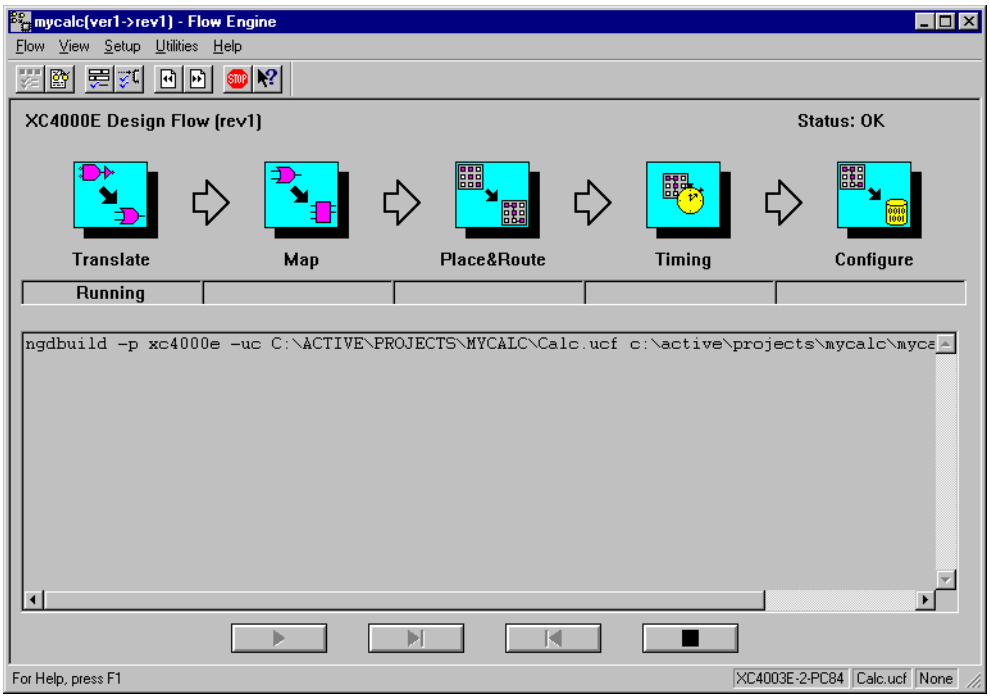


Figure 4-58 Flow Engine

The status bar shows the progress of the implementation flow with the following stages:

- Translate: converts the design EDIF file into an NGD (Native Generic Design) file.
- Map: groups basic elements such as flip-flops and gates into logic blocks (“comps”); also generates a logic-level timing report if selected.
- Place&Route: places comps into the device, and routes the signals between them.
- Timing: generates timing simulation data and the post-layout timing report.
- Configure: generates a bitstream suitable for downloading into a device.

When implementation is complete, an Implementation Status box appears with the following message:

```
Implementing revision ver1->rev1 completed
successfully.
```

9. Click OK in the Implementation Status dialog to return to the Design Manager. Note that the status of rev1 is (Implemented, OK). This means that the revision has reached the Implemented state with no errors.
10. Select **File** → **Exit** to close the Design Manager. Click Yes to confirm that you wish to exit.

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.

Invoking the Logic Simulator for Timing Simulation

1. Click on the SIM Timing button in the Project Flowchart. The timing simulation netlist that was created by the Design Implementation Tools is loaded into the Logic Simulator.

   Note that the list box in the toolbar reads `Timing`. This indicates that the simulator is operating in Timing mode.

2. Select **`File`** → **`Load Waveform`**. In the dialog box, select calcsim1.tve and click OK.

   The signals and stimulators you saved during functional simulation are loaded into the Waveform Viewer. The simulation mode is also changed back to Functional mode.

3. Select timing from the drop-down list box at the end of the horizontal toolbar.

Note: The Logic Simulator can operate in two other modes, which are not used for most designs. Refer to the online help for information about simulation modes.

Asserting Global Reset

The global reset signal must be pulsed at the beginning of all timing simulations. This signal sets or resets all flip-flops in the chip. Whether a flip-flop is set or reset depends on the target device family, and whether it is an FDPE or an FDCE flip-flop. The default configuration for all flip-flops is to function as a reset flip-flop.

Without STARTUP

Note: This section applies to designs in which the STARTUP block has not been used. If you have an XC4000 family design that has the STARTUP block in it, go to the "With STARTUP (XC4000 Family Only)" section.

This signal does not exist on the schematic, but it does exist in the device and in the timing simulation netlist. The name and polarity of the global reset signal depends on the target device family, as shown in the following table.

Table 4-3 Global Reset Signals

Dev. Family	Net Name	Polarity
XC4000	GSR	Active-High
XC9500	PRLD	Active-High

1. Add the GSR (or PRLD) signal to the Waveform Viewer.
2. Select **`Signal`** → **`Add Stimulators`**.
3. Using the procedure described in the "Defining Formulas" section, assign the following formula to the F2 stimulator:

   `H100L3000`
4. In the Waveform Viewer, select the GSR (or PRLD) signal.
5. In the Stimulator Selection window, click on the F2 LED.
6. Click Close.

This will pulse the global reset signal high for 100 ns, then drive it low for the remainder of the simulation.

With STARTUP (XC4000 Family Only)

Note: This section applies to designs in which the STARTUP block has been used. If you have a design without a STARTUP block, follow the instructions in the "Without STARTUP" section.

The global reset signal in the XC4000 family is not hard-wired to a package pin and need not appear on one at all. If you want access to the global reset net from an external pin, place the STARTUP component in your schematic and attach an IPAD and IBUF to the GSR pin.

This pad becomes an active-High global set / reset signal. You can also use an internally generated signal to drive the GSR pin of the STARTUP component. There is also an active-High Global Three State signal (GTS) that you can access in the same way. See the online *Libraries Guide* for more information on the STARTUP symbol.

1. Add the NOTGBLRESET signal to the Waveform Viewer.
2. Select **Signal** → **Add Stimulators**.
3. Using the procedure described in the "Defining Formulas" section, assign the following formula to the F2 stimulator:

   L100H3000

4. In the Waveform Viewer, select the NOTGBLRESET signal.
5. In the Stimulator Selection window, click on the F2 LED.
6. Click Close.

This formula pulses the NOTGBLRESET signal low for 100 ns, then drives it high for the remainder of the simulation. Note that, because of the inverter in the path from NOTGBLRESET to the GSR pin of the STARTUP block, this signal is active-Low.

Running the Simulation

The simulation preferences, options settings, and zoom level are saved from the last time you used the Logic Simulator.

1. Click on the Long button three times and the Step button once.

   You may need to zoom in on the waveforms to see the timing delays. You can quickly zoom in on an area by clicking on the time ruler and dragging over the period you wish to zoom in on.

2. Click and drag over the time period near 1 us, where the ALUVAL3 bus changes from 1 to E.
3. To measure the delay between the rising edge of CLK and the change on the ALUVAL3 bus, select **Waveform** → **Measurements** → **Measurements On**. The cursor changes into a double headed horizontal arrow, with a vertical arrow on the left.
4. Position the vertical arrow beneath the rising edge of the CLK signal and click to mark the beginning of the measurement. The vertical arrow moves to the right.

5. Click on the ALUVAL3 bus transition to complete the measurement. A green line appears to mark the measurement.
6. Select **Waveform** → **Measurements** → **Measurements On** again to exit the Waveform Measurement mode. You may need to zoom in further to see the measurement value displayed in the Waveform Viewer.

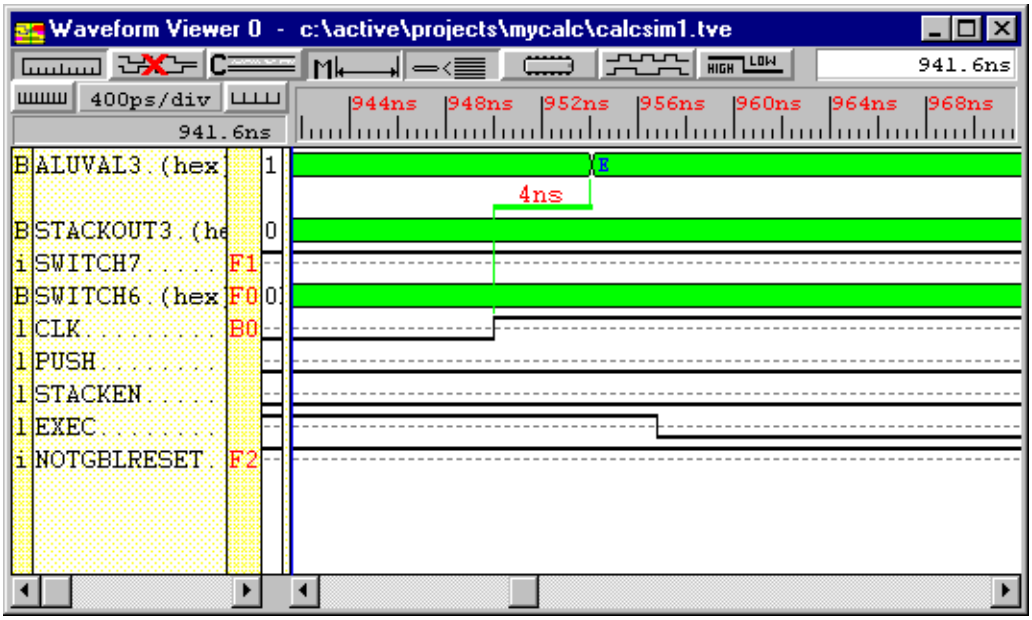


Figure 4-59 Measuring Delays

7. After examining the waveforms, exit the Logic Simulator.

Examining Routed Designs with EPIC

Note: This section applies only to FPGA designs. If you are targeting a CPLD such as an XC9000 device, skip to the "Making Incremental Design Changes" section.

At this point in the tutorial, the design process is complete. If you would like to see how the design has been implemented, you can take a graphical look at your placed and routed design using the Editor for Programmable Integrated Circuits, (EPIC). You can access EPIC from the toolbar in the Design Manager.

EPIC provides several useful functions, such as

- manual placement of a pre-routed design
- manual editing of a routed design
- static timing analysis

EPIC is explained in a separate tutorial. See the "EPIC Tutorial" chapter of the *EPIC Reference/User Guide*. Before starting this tutorial, be sure to select the ver1 → rev1 revision of the design in the project view.

Verifying the Design Using a Demonstration Board

Note: This section applies only to FPGA designs. If you are targeting a CPLD such as an XC9000 device, skip to the "Making Incremental Design Changes" section.

A bitstream has been created during the Configure stage in the Flow Engine. At this point, you are ready to download the bitstream using a parallel download cable or the more versatile XChecker cable connected to your PC. The XC4000E version of the Calc design is suitable for download into an FPGA demonstration board available from Xilinx.

Downloading is accomplished with the Hardware Debugger. To invoke the Hardware Debugger, select **`Tools`** → **`Hardware Debugger`** from the menu bar, or click on the Hardware Debugger button in the toolbar. If you are using an XChecker cable, you can also use the Hardware Debugger to read back information from the device to verify both the configuration as well as the state of memories and registers within the device.

The Hardware Debugger is explained in a separate tutorial. See the "CALC Tutorial" chapter of the *Hardware Debugger Reference/User Guide*. Before starting this tutorial, be sure to select the ver1 → rev1 revision of the design in the project view.

Making Incremental Design Changes

After initially placing and routing a design, it is often necessary to go back to the schematic and make slight modifications to the original design. When this situation occurs, much of the place-and-route information from the previous design iteration can be reused, as much of it is unchanged. This process is known as incremental design, and the NCD file (containing partition, placement, and routing information) from the prior place-and-route iteration is called the guide file.

Since much of the place-and-route information is extracted from the guide file, the place-and-route time is greatly reduced. The reuse of place-and-route information also results in more stable timing over a number of guided place-and-route iterations. Once a section of your design passes your timing requirements, guided design ensures that it will pass in the future, even if other parts of the design are modified.

In this section of the tutorial, you make a small change to the schematic and reprocess the design using the guide option in the mapping program (MAP) and the place-and-route program (PAR).

Note: A small design change is the addition, removal, or replacement of only a small amount of logic in the design; the exact amount is dependent on the size of the design. If radical changes are made to a design, especially changes to existing portions of the design, it may be disadvantageous to guide the design.

Making an Incremental Schematic Change

Make a simple change to the Calc schematic that will be visible immediately on the demonstration board. For example, assume that the reset opcode is no longer needed and needs to be removed form the design. This can be done by grounding the ‘R’ pins of the FDRE and FD4RE macros in the ALU schematic. The logic that generated the original reset signal, and the logic it drove, is automatically optimized out of the netlist by the MAP program.

1. In the Project Manager, expand the hierarchy view by clicking on the icon next to calc.sch.
2. Double click on ALU to start the Schematic Editor.
3. Zoom in on the lower right quadrant of the schematic.

4. Select the AND5B2 component that generates the QRESET net feeding the FDRE and FD4RE.
5. Select **Edit** → **Delete** or press Del to delete the component.
6. Connect a GND symbol to the R pin of the FD4RE symbol, as shown in the figure below.
7. Save the schematic and exit the Schematic Editor.

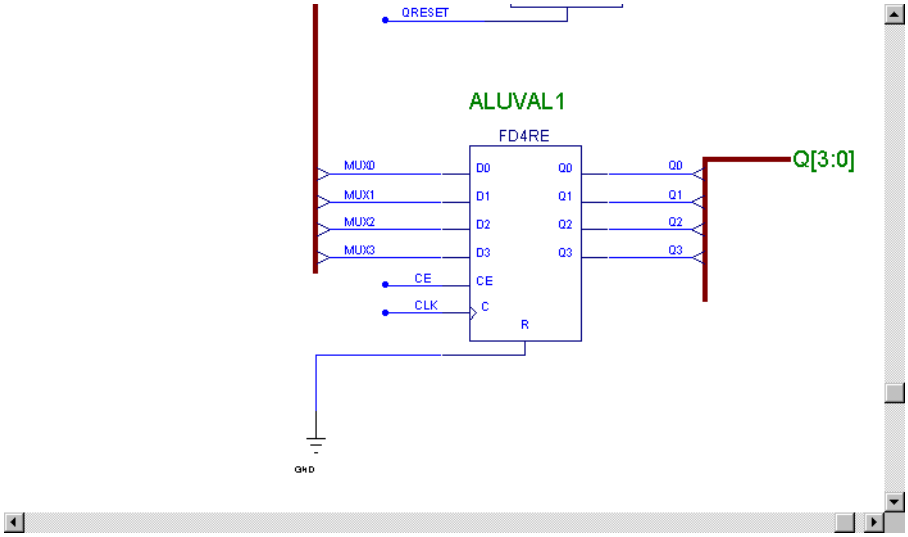


Figure 4-60 Grounding the Reset Logic

Translating the Incremental Design

Translate the guided Calc design by turning on the guide options in Flow Engine. The following instructions demonstrate an alternative method of running Flow Engine that offers more control over the implementation flow.

1. In the Xilinx Design Manager, select calc, then choose **Design** → **New Version**.
2. The New Version dialog box appears with the Name field automatically filled in as "ver2". You may also add a comment to the new version. This comment appears in the project view next to the version number. Click OK.

Note: You can add a comment to any version or revision in the project view by selecting that version or revision, then selecting **Design** → **Properties**.

3. Select the newly created "ver2" in the project view, then select **Design** → **New Revision**.
4. The New Revision dialog box appears with the Name field automatically filled in as "rev1" and the Part field automatically filled in as "XC4003E-4-PC84". You may add a comment to the new revision if you wish. Click OK.
5. Select the newly created "rev1" in the project view, then select **Tools** → **Flow Engine**. Alternatively, you can click the Flow Engine button in the Toolbox.

   The Flow Engine appears. However, unlike the procedure you used in the first revision, the implementation flow does not start automatically. This allows you to step forward and even backward through the implementation flow by individual stages, using the buttons at the bottom of the Flow Engine window or the selections underneath the Flow menu.

6. Select **Setup** → **Options** from the menu bar. The Options dialog box appears as before.
7. Go through the different options as before and verify that the settings are the same as in the previous revision.

8. In the Guide Design field, select Last. This sets the previous revision of the placed and routed design. In this case, it has the same effect as selecting ver1 → rev1.
9. Click OK to return to Flow Engine.
10. Run the implementation as before by clicking the Run icon (on the far left) at the bottom on the Flow Engine window.
11. When all steps have completed successfully, select **`Flow`** → **`Close`** to exit Flow Engine.

Verifying the Change in the Demonstration Board

Verify that the change was performed by downloading the new bitstream to the demonstration board, as you did previously. As before, see the "CALC Tutorial" chapter of the *Hardware Debugger Reference/User Guide* for more information. Before running through this tutorial, make sure that the ver2 → rev1 revision is selected in the project view.

Further Reading

This tutorial has given you the information necessary to complete a typical design cycle using the 1.4 version of Foundation. There are many commands and options available within both the design entry tools and the design implementation tools that are not covered in this tutorial. Refer to the online help files and the online manuals (viewable with the DynaText browser) for complete documentation of all the features in this release.

Appendix A

Glossary

This appendix contains definitions and explanations for terms used in the *Foundation Series Quick Start Guide 1.4*.

aliases

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

attribute

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

BLD file

The translation report that 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.

block

A group consisting of one or more logic functions. Also called CLB.

component

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

constraint

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

Using constraints, 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.

design entry tools

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

An optional package, Foundation Express, contains the VHDL and Verilog design languages.

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, PROMGen, and EPIC.

EDIF2NGD

EDIF2NGD reads the constraints in an NCF (netlist constraints file) file and adds the constraints to the output NGO file.

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.

HDL

HDL (hardware description language).

LCA file

An LCA file is a mapped file of a Xilinx design produced by an earlier software release.

LCA2NCD

LCA2NCD converts an LCA file to an NCD file. The NCD file produced by LCA2NCD can be placed and routed, viewed in EPIC, analyzed for timing, and back-annotated.

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 is used for data manipulation and control functions that require higher speed than a microprocessor can provide.

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, and it can also be unrouted, partially routed, or fully routed. See also “physical macro.”

MCS file

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

MDF file

An MDF (MAP directive file) file is a file describing how logic was decomposed when the design was originally mapped. The MDF file is used for guided mapping using Xilinx Development System software. The MDF file enables the MAP program to recreate the decompositions chosen when the design was first mapped.

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 EPIC. 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.

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.

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.

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 an output file of the MAP program. It is an ASCII file containing physical constraints created by the MAP program as well as physical constraints entered by you. You can edit the PCF file from within EPIC.

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 EPIC, BitGen, PAR, and Hardware Debugger. By default, results of the DRC are written into the current working directory.

physical macro

A physical macro is a logical function that has been created from components of a specific device family. Physical macros are stored in files with the extension .nmc. A physical macro is created when EPIC is in macro mode. See also “macro.”

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).

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 EPIC. Route-throughs provide you with routing resources that would otherwise be unavailable.

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.

TRCE

TRCE (Timing Reporter and Circuit Evaluator) “trace” is a program that will automatically perform a timing analysis on a design using available timing constraints. The input to TRCE is a mapped 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.

(Historical note: TRCE should not be confused with the UNIX trace command. The UNIX trace command is used to trace system calls and signals).

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.

wire

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

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Z