XAPP473 (v1.1) May 23, 2005

Using the ISE Design Tools for Spartan-3 Generation FPGAs

Summary

Software is critical to the effective use of programmable logic. The Spartan™-3 Generation is supported by the complete set of Xilinx Integrated Software Environment (ISE) development tools, with additional support available from a variety of partners. This document provides an overview of those design tools. It is intended primarily for the user who is new to the Xilinx development system. This document can be used to get a better understanding of the specific tools mentioned elsewhere in the Spartan-3 Generation literature. The first half provides an overview of the general design flow, while the second half describes the specific tools used at the different steps in the flow. Use the Xilinx development system documentation for detailed information and introductory tutorials.

This application note applies to all Spartan™-3 Generation FPGA families, which include the Spartan-3 family, the Spartan-3L family, and the Spartan-3E family.

Introduction

Combined with the Spartan-3 Generation FPGA family, ISE's optimized design tools help you finish faster and lower your project costs. The ISE package is a collection of Xilinx software design tools that concentrate on delivering the most productivity available for your Spartan-3 Generation logic performance. With ProActive Timing Closure technology, you get the fastest runtimes in programmable logic ensuring you reach your performance goals quicker. Incremental Design delivers faster re-compile times with guaranteed performance, and the optional Xilinx ChipScope™ Pro verification tools provide real-time debug with advantages that are not possible in ASIC designs. ISE makes sure you get through the logic design process faster, saving both time and project costs, and getting you to market ahead of your competition.

Design Flow

The standard design flow for Spartan-3 Generation FPGAs consists of the following three major steps. The entire design implementation flow is run simply by selecting the desired result in the Xilinx Graphical User Interface (GUI). The tools automatically determine which programs and files are needed to bring the appropriate output up to date.

1. Design Entry and Synthesis

   In this step of the design flow, you create your design using a Xilinx-supported schematic editor, a Hardware Description Language (HDL) for text-based entry, or both. If you use an HDL for text-based entry, you must synthesize the HDL file into an industry-standard Electronic Data Interchange Format (EDIF) file. If you use the Xilinx Synthesis Technology (XST) tool, a Xilinx-specific NGC netlist file is created, which can be converted to an EDIF file.

2. Design Implementation

   By implementing the specific Xilinx Spartan-3 Generation architecture, you convert the logical design file format, such as EDIF, that you created in the design entry or synthesis stage into a physical file format. The physical information is contained in the Native Circuit Description (NCD) file. Then you create a bitstream file from these files and optionally program a PROM for subsequent programming of your Spartan-3 Generation device.

3. Design Verification

   Using a gate-level simulator, you ensure that your design meets your timing requirements and functions properly. In-circuit verification can be performed by downloading your design

© 2005 Xilinx, Inc. All rights reserved. XILINX, the Xilinx logo, and other designated brands included herein are trademarks of Xilinx, Inc.
All other trademarks are the property of their respective owners.

to the device using Xilinx iMPACT Programming Software. Design verification can begin immediately after design entry and can be repeated after various steps of design implementation.

Figure 1 shows the general overall design flow for Spartan-3 Generation FPGAs.

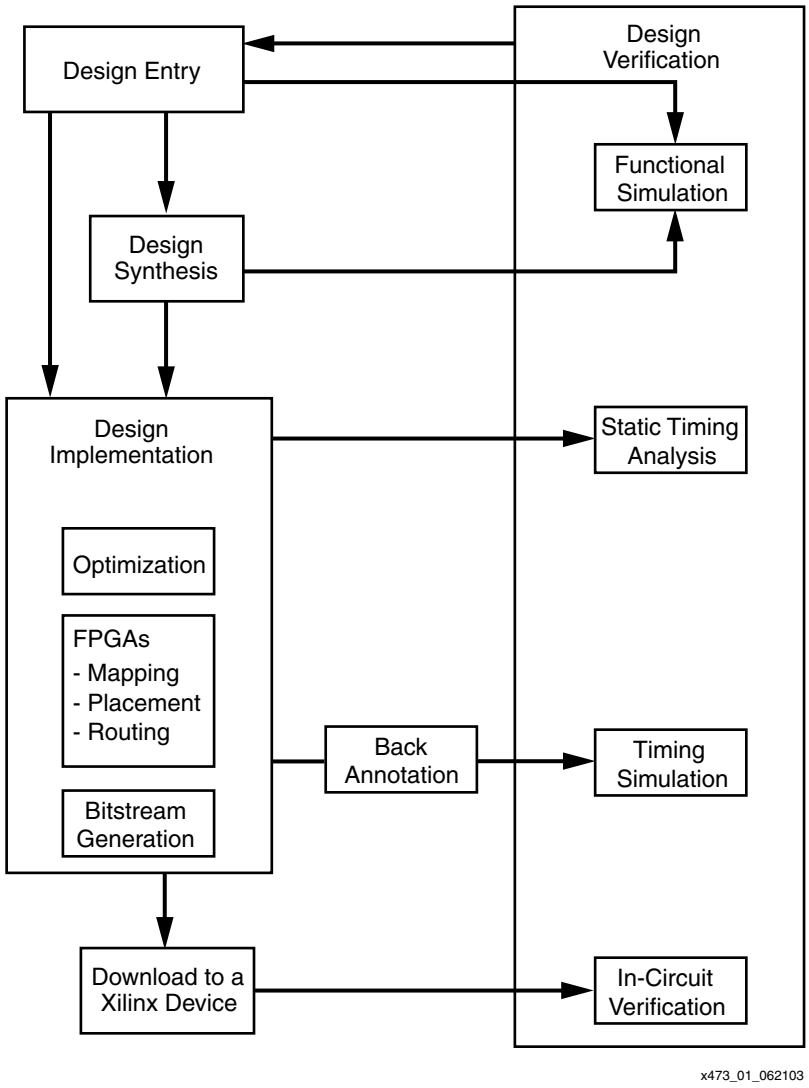


Figure 1: **Design Flow**

Design Entry and Synthesis

You can enter a design with a schematic editor or a text-based tool for HDL code. Design entry begins with a design concept, expressed as a drawing or functional description. From the original design, a generic EDIF netlist is created, then synthesized and translated into a Xilinx netlist file. This file is fed into a program called NGDBuild, which produces a logical Native Generic Database (NGD) file. Xilinx libraries provide access to features specific to the Spartan-3 Generation architecture.

Figure 2 shows the design entry and synthesis flow.

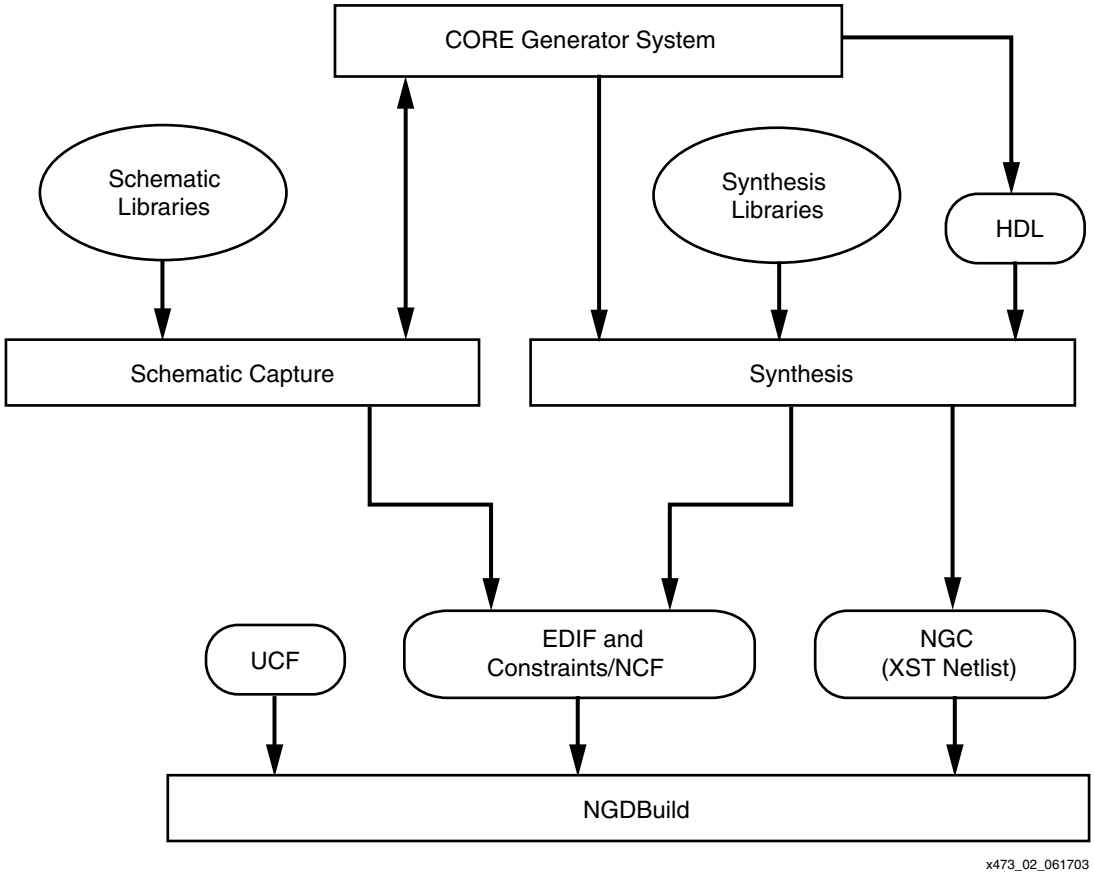


Figure 2: **Design Entry and Synthesis Flow**

Hierarchical Design

Design hierarchy is important in both schematic and HDL entry for the following reasons:

- Helps you conceptualize your design
- Adds structure to your design
- Promotes easier design debugging
- Makes it easier to combine different design entry methods (schematic, HDL, or state editor) for different parts of your design
- Makes it easier to design incrementally, which consists of designing, implementing, and verifying individual parts of a design in stages
- Reduces optimization time
- Facilitates concurrent design, which is the process of dividing a design among a number of people who develop different parts of the design in parallel, such as in Modular Design

Xilinx strongly recommends that you name the components and nets in your design. These names are preserved and used by the Xilinx tools. These names are also used for back-annotation and appear in the debug and analysis tools. If you do not name your components and nets, the tools automatically generate the names, making it difficult to analyze circuits.

Schematic Entry

Schematic tools provide a graphical interface for design entry. You can use these tools to connect symbols representing the logic components in your design. You can build your design with individual gates, or you can combine gates to create functional blocks.

Primitives and macros are the “building blocks” of a device library. The Xilinx Spartan-3 Generation library provides primitives as well as common high-level macro functions, all optimized for the Spartan-3 Generation architecture. Primitives are basic circuit elements, such as AND and OR gates, and special device resources, such as the DCM and block RAM. Each primitive has a unique library name, symbol, and description.

Macros contain multiple library elements, which can include primitives and other macros. Soft macros have pre-defined functionalities, but have flexible mapping, placement, and routing. Relationally Placed Macros (RPMs) have fixed mapping and relative placement. Macros are not available for synthesis because synthesis tools have their own module generators and do not require RPMs. If you wish to override the module generation, you can instantiate Xilinx-provided CORE Generator™ modules, which include pre-built optimization for the Spartan-3 Generation architecture. For most leading-edge synthesis tools, this is not needed unless it is for a module that cannot be inferred.

HDL Entry and Synthesis

A typical Hardware Description Language (HDL) supports a mixed-level description in which gate and netlist constructs are used with functional descriptions. This mixed-level capability enables you to describe system architectures at a high level of abstraction, then incrementally refine a design’s detailed gate-level implementation. HDL descriptions offer the following advantages:

- You can verify design functionality early in the design process. A design written as an HDL description can be simulated immediately. Design simulation at this high level — at the gate-level before implementation — allows you to evaluate architectural and design decisions.
- An HDL description is more easily read and understood than a netlist or schematic description. HDL descriptions provide technology-independent documentation of a design and its functionality. Because the initial HDL design description is technology independent, you can use it again to generate the design in a different technology, without having to translate it from the original technology.
- Large designs are easier to handle with HDL tools than schematic tools.

After creating your HDL design, you must synthesize it. During synthesis, behavioral information in the HDL file is translated into a structural netlist, and the design is optimized for the Spartan-3 Generation architecture. Xilinx supports HDL synthesis tools for several third-party synthesis vendor partners. In addition, Xilinx offers its own synthesis tool, Xilinx Synthesis Technology (XST).

Functional simulation tests the logic in your design to determine if it works properly. You can save time during subsequent design steps if you perform functional simulation early in the design flow.

Although HDL entry offers the advantage of technology independence, it is helpful to understand the available resources in the Spartan-3 Generation architecture and design to take advantage of those resources. For example, the abundance of registers at every I/O and following every look-up table encourages pipelining. Most synthesis tools automatically infer Xilinx-specific resources and optimize for the architecture. Simple ways to specify implementation requirements are to instantiate Spartan-3 Generation library components or add constraints.

Constraints

You might want to constrain your design within certain timing or placement parameters to specify your required pin locations or timing requirements. You can specify logic mapping, block placement, and timing specifications. Constraints can be entered as parameters or attributes on library components. You can enter constraints by hand or use one of several graphical tools for generating constraint files and evaluating the results. Constraints found in the design are

written to an NCF file (Netlist Constraints File). Constraints created separately are written to a UCF file (User Constraints File).

Design Implementation

Design Implementation begins with the translating and then mapping of a logical design file to a specific Spartan-3 Generation device. It is complete when the physical design is successfully routed and a bitstream is generated. You can alter constraints during implementation in the same way as during the Design Entry step.

Figure 3 shows an overall view of the design implementation flow for Spartan-3 Generation FPGAs.

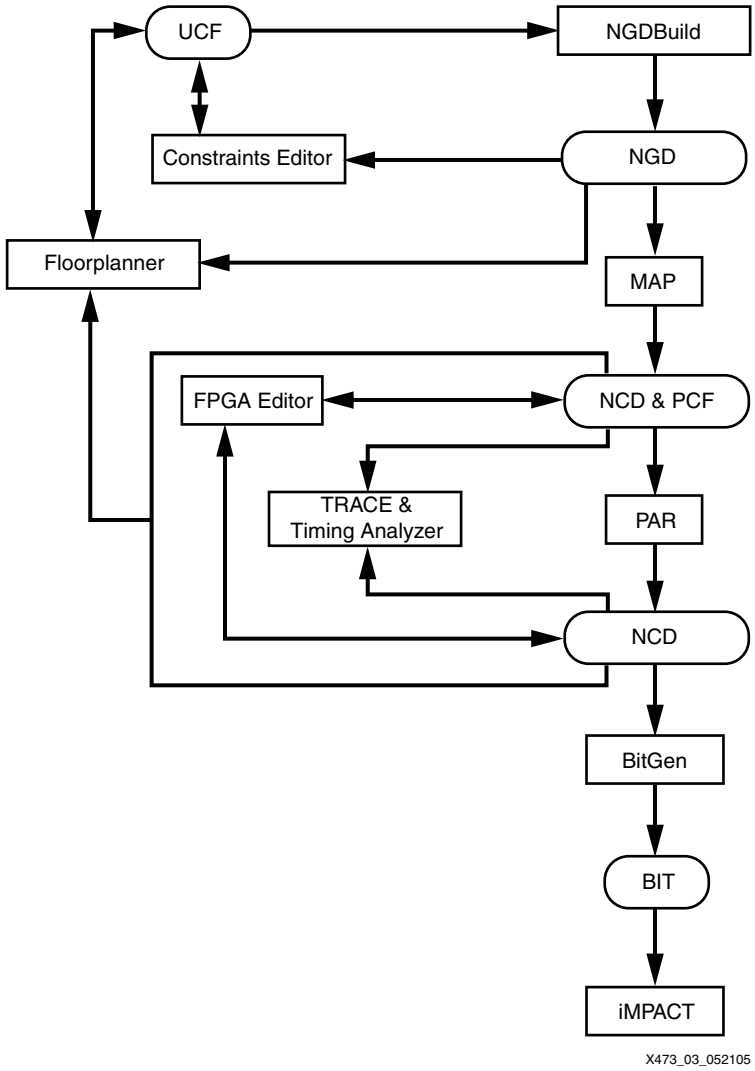


Figure 3: **Design Implementation Flow**

Translating

NGDBuild performs all the steps necessary to read a netlist file in EDIF or NGC format and create an NGD file describing the logical design. A logical design is in terms of logic elements, such as AND gates, OR gates, decoders, flip-flops, and RAMs. The NGD file resulting from an NGDBuild run contains both a logical description of the design reduced to Xilinx primitives and a description in terms of the original hierarchy expressed in the input netlist. The output NGD file then can be mapped to the Spartan-3 device family resources.

NGDBuild performs the following steps to convert a netlist to an NGD file:

1. Reads the source netlist(s). NGDBuild invokes the Netlister Launcher. The Netlist Launcher determines the type of the input netlist and starts the appropriate netlist reader program. The netlist readers incorporate NCF files associated with each netlist. NCF files contain timing and layout constraints for each module.
2. Reduces all components in the design to NGD primitives. NGDBuild merges components that reference other files. NGDBuild also finds the appropriate system library components, physical macros, and behavioral models.
3. Checks the design by running a Logical Design Rule Check (DRC) on the converted design. The Logical DRC is a series of tests on the logical design.
4. Writes an NGD file as output.

Mapping

The MAP program maps a logical design to a Spartan-3 Generation FPGA. The input to MAP is an NGD file, which contains a logical description of the design in terms of both the hierarchical components used to develop the design and the lower-level Xilinx primitives. Additionally, it contains any number of hard placed-and-routed physical macro files. MAP then maps the logic to the components (logic cells, I/O cells, and other components) in the Spartan-3 Generation architecture. The output design is a Native Circuit Description (NCD) file, which is a physical representation of the design mapped to the components in the Spartan-3 Generation architecture. The NCD file then can be placed and routed.

MAP performs the following steps when mapping a design:

1. Selects the target Xilinx device, package, and speed.
2. Reads the information in the input design file.
3. Performs a Logical DRC (Design Rule Check) on the input design. If any DRC errors are detected, the MAP run is aborted. If any DRC warnings are detected, the warnings are reported, but MAP continues to run.
4. Removes unused logic, where all unused components and nets are removed.
5. Maps pads and their associated logic into IOBs.
6. Maps the logic into Xilinx components (IOBs, CLBs, etc.). If any Xilinx mapping control symbols appear in the design hierarchy of the input file, MAP uses the existing mapping of these components in preference to re-mapping them. The mapping is influenced by various constraints.
7. Updates the information received from the input NGD file and writes this updated information into an NGM file. This NGM file contains both logical information about the design and physical information about how the design was mapped. The NGM file is used only for back-annotation.
8. Creates a physical constraints (PCF) file. This text file contains any constraints specified during design entry. If no constraints were specified during design entry, an empty file is created so that you can enter constraints directly into the file using a text editor.
9. Runs a physical Design Rule Check (DRC) on the mapped design. If DRC errors are found, MAP does not write an NCD file.
10. Creates an NCD file, which represents the physical design. The NCD file describes the design in terms of Xilinx components (CLBs, IOBs, and so forth).
11. Writes a MAP report (MRP) file, which lists any errors or warnings found in the design, details how the design was mapped, and supplies statistics about component usage in the mapped design.

Placing and Routing

After creating a mapped NCD file, you can place and route the file using the automatic Place And Route (PAR) tool. PAR accepts an NCD file as input, places and routes the design, and outputs an NCD file to be used by the bitstream generator (BitGen). You can use the output NCD file as a guide file for additional runs of PAR after making minor changes to your design.

PAR places and routes a design based on the following considerations:

- Cost-Based: Placement and routing are performed using various cost tables that assign weighted values to relevant factors such as constraints, length of connection, and available routing resources.
- Timing-Driven: The Xilinx timing analysis software enables PAR to place and route a design based upon your timing constraints.

Placing

The PAR placer executes multiple phases of the placer. PAR writes the NCD after all the phases are completed. During placement, PAR places components into sites based on factors such as constraints specified in the PCF file, the length of connections, and the available routing resources. Timing-driven placement is automatically invoked if PAR finds timing constraints in the physical constraints file.

Routing

The next stage is routing the placed design. PAR writes the NCD file when the design is fully routed. There still may exist unroutes since power and ground are not considered. At this point the design can be analyzed against timing. A new NCD is written as the routing improves. The router performs a procedure to converge on a solution that routes the design to completion and meets timing constraints. Timing-driven routing is automatically invoked if PAR finds timing constraints in the physical constraints file.

Floorplanning

Floorplanning is the process of specifying user placement constraints. The Floorplanner provides a graphical view of placement, while the FPGA Editor provides a graphical view of both placement and routing. Both tools can be used before or after PAR to analyze or constrain the design.

Bitstream Generation

After the design has been completely routed, it is necessary to configure the device so that it can execute the desired function. This configuration is done using files generated by BitGen, the Xilinx bitstream generation program. BitGen takes a fully routed NCD file as its input and produces a configuration bitstream (binary BIT file).

The BIT file contains all of the configuration information from the NCD file defining the internal logic and interconnections of the Spartan-3 Generation FPGA, plus device-specific information from other files associated with the target device. The binary data in the BIT file then can be downloaded into the FPGA memory cells or it can be used to create a PROM file.

Design Verification

Design verification is the process of testing the functionality and performance of your design. You can verify Xilinx designs in the following ways:

- Simulation (functional and timing using back-annotation)
- Static timing analysis
- In-circuit verification

Design verification procedures should occur throughout your design process, as shown in Figure 4.

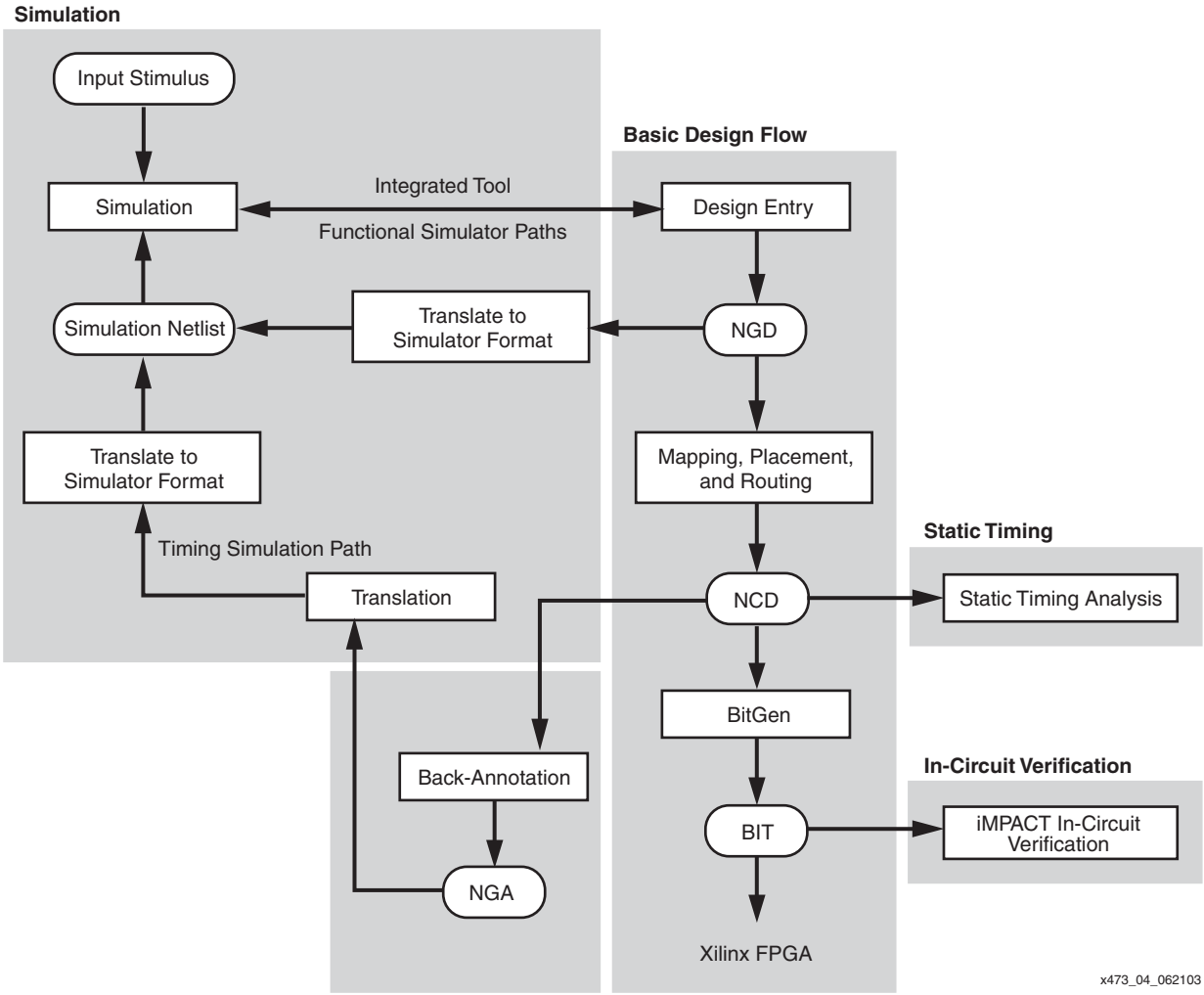


Figure 4: **Design Verification Flow**

Simulation

Design simulation involves testing your design using software models. It is most effective when testing the functionality of your design and its performance under worst-case conditions. You can easily probe internal nodes to check your circuit's behavior, and then use these results to make changes in your design. Simulation is performed using Xilinx or third-party tools that are linked to the Xilinx Development System. The software models provided for your simulation tools are designed to perform detailed characterization of your design. You can perform functional or timing simulation.

Functional Simulation

Functional simulation determines if the logic in your design is correct before you implement it in a device. Functional simulation can take place at the earliest stages of the design flow. Because timing information for the implemented design is not available at this stage, the simulator tests the logic in the design using unit delays.

Timing Simulation

Timing simulation verifies that your design runs at the desired speed for your device under worst-case conditions. This process is performed after your design is mapped, placed, and

routed. At this time, all design delays are known. Timing simulation is valuable because it can verify timing relationships and determine the critical paths for the design under worst-case conditions. It also can determine whether or not the design contains setup or hold violations. Before you can simulate your design, you must go through the back-annotation process, as described below. During this process, the Xilinx netlist writers create suitable formats for various simulators.

Note that naming the nets during your design entry is important for both functional and timing simulation because it allows you to find the nets in the simulations more easily than looking for a software-generated name.

Back-Annotation

Before timing simulation can occur, the physical design information must be translated and distributed back to the logical design. This back-annotation process is done with a program called NGDAnno. These programs create a database for the netlist writers, which translate the back-annotated information into a netlist format that can be used for timing simulation.

NGDAnno is a command line program that distributes information about delays, setup and hold times, clock to out, and pulse widths found in the physical NCD design file back to the logical NGD file. NGDAnno reads an NCD file as input. The NCD file can be a mapped-only design, or a partial or fully placed and routed design. An NGM file, created by MAP, is an optional source of input. NGDAnno merges mapping information from the NGM file with placement, routing, and timing information from the NCD file. NGDAnno outputs a Native Generic Annotated (NGA) file, which is a back-annotated NGD file. This file is input to the appropriate netlist writer, which converts the binary Xilinx database format back to an ASCII netlist.

Netlist Writers (NGD2EDIF, NGD2VER, or NGD2VHDL) take the output of NGDAnno and create a simulation netlist in the specified format. An NGD or NGA file is input to each of the netlist writers. The NGD file is a logical design file containing primitive components, while the NGA file is a back-annotated logical design file.

Static Timing Analysis

Static timing analysis is best for quick timing checks of a design after it is placed and routed. It also allows you to determine path delays in your design. Following are the two major goals of static timing analysis:

- Timing verification is the process of verifying that the design meets your timing constraints.
- Reporting is the process of enumerating input constraint violations and placing them into an accessible file. You can analyze partially or completely placed and routed designs. The timing information depends on the placement and routing of the input design.

You can run static timing analysis using the Timing Reporter And Circuit Evaluator (TRACE) program, which is accessible through the Timing Analyzer GUI. Use either tool to evaluate how well the place and route tools met the input timing constraints.

In-Circuit Verification

As a final test, you can verify how your design performs in the target application. In-circuit verification tests the circuit under typical operating conditions. Because you can program your Xilinx devices repeatedly, you can easily load different iterations of your design into your device and test it in-circuit. To verify your design in-circuit, download your design bitstream into a device using the iMPACT programming software with the Parallel Cable IV, MultiPRO Desktop Tool, or Platform Cable USB.

ISE Development Environment

Introduction to ISE

Xilinx development systems are available in a number of easy to use configurations, collectively known as the Integrated Software Environment (ISE) Series. Creating Spartan-3 Generation designs is easy with Xilinx ISE development systems, which support advanced design capabilities, including ProActive Timing Closure, integrated logic analysis, and the fastest place and route runtimes in the industry. ISE solutions enable designers to get the performance they need, quickly and easily.

Note: To get the full details on ISE tools for Spartan-3 Generation devices, go to http://www.xilinx.com/ise/ise_promo/ise_spartan3.htm.

Project Navigator is the user interface that helps you manage the entire design process including design entry, simulation, synthesis, implementation and finally configuration of your device.

The following is an outline of the features offered in ISE:

Design Entry

- HDL Editor
- StateCAD® State Machine Editor
- Schematic Editor - Engineering Capture System (ECS)
- CORE Generator system

Synthesis

- XST - Xilinx Synthesis Technology
- Integration with LeonardoSpectrum synthesis from Mentor Graphics
- Integration with Synplify/Pro and Amplify synthesis from Synplicity

Simulation

- HDL Bencher™ Testbench Generator
- Integration with ModelSim Simulator from Model Technology

Implementation

- Translate
- Map
- Place and Route (PAR)
- Floorplanner
- FPGA Editor
- Timing Analyzer
- XPower Power Analysis

Device Download

- BitGen Bitstream Generator
- iMPACT Configuration Tool
- ChipScope Pro Logic Analyzer

ISE Versions

The ISE development systems are available in the following configurations.

- ISE WebPACK™ Tool

  The ISE WebPACK tool is the easiest development system to get. This free tool is downloadable from the Web at: (http://www.xilinx.com/xlnx/xil_prodcat_landingpage.jsp?title=ISE+WebPack).

  ISE WebPACK software combines support for advanced HDL entry, synthesis, and verification capabilities for all Xilinx CPLDs and lower-density FPGAs.

- ISE BaseX Tool

  The ISE BaseX tool is the industry's most cost-effective, PC-based programmable logic design environment. The ISE BaseX configuration provides all of the capabilities contained within the ISE WebPACK software plus additional tools like the CORE Generator system and FPGA Editor that help you complete your programmable logic design even faster.

- ISE Alliance Tool

  The ISE Alliance tool is designed to fit into your existing design environment. This tool provides full device support and works seamlessly with those tools of our EDA partners. The ISE Alliance tool is for those designers looking to add programmable logic design capabilities into their existing design environment. This tool does not include XST synthesis or ECS schematic capture.

- ISE Foundation™ Tool

  The ISE Foundation tool is a complete, ready-to-use design environment that integrates schematic, synthesis, and verification technologies into an intuitive, yet highly advanced design solution. The tool has full device support as well as the full suite of tools.

To see a table comparison of these versions, see the Development Systems Overview at http://www.xilinx.com/ise/devsys_feature_guide.pdf.

Development system updates are provided on a regular basis. These are available as Service Packs that can be downloaded from the Xilinx website (http://www.xilinx.com/support/software/install_info.htm). Always use the latest development system update for the best results.

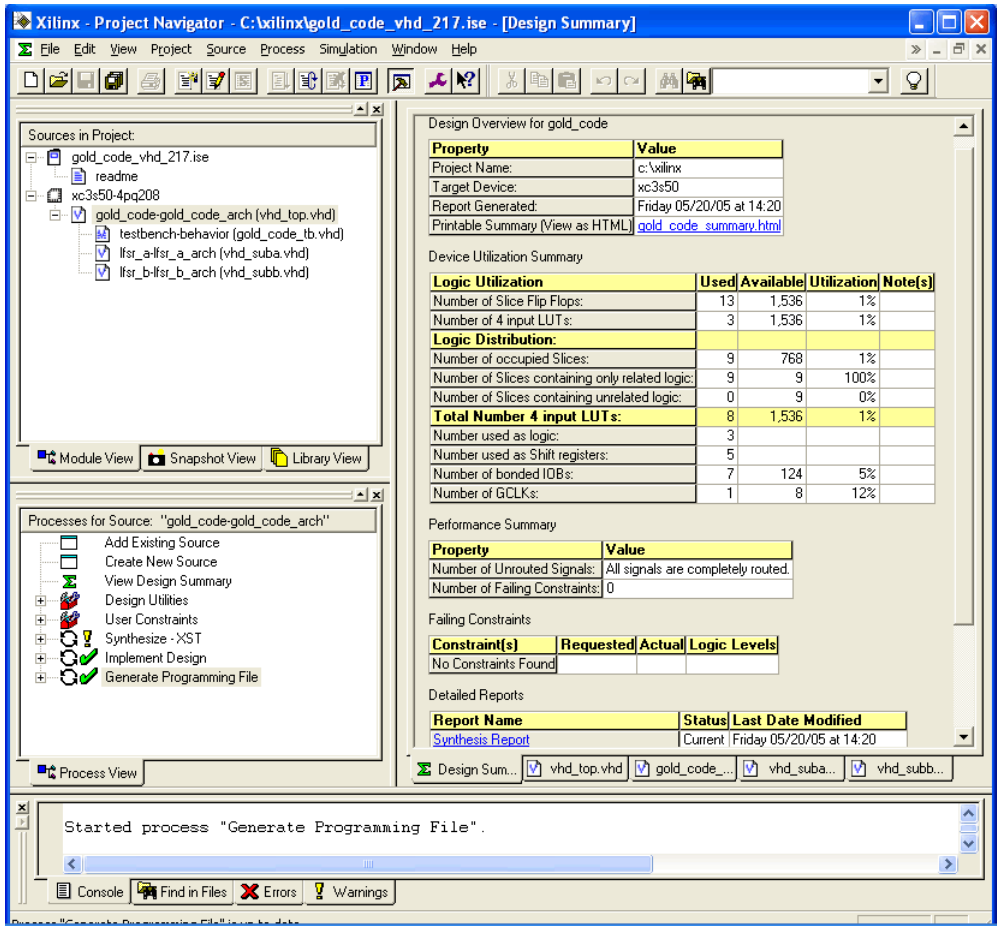
Project Navigator

Project Navigator is the primary user interface for the Xilinx ISE tools. You can create, define, and compile your Spartan-3 Generation design using a suite of tools accessible from Project Navigator. Each step of the design process, from design entry to downloading the design to the device, is managed from Project Navigator as part of a project. These include:

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

Project Navigator Main Window

The Project Navigator workspace is made up of a title bar, a status bar, a menu bar, toolbars and windows.



X473_05_052105

Figure 5: **Project Navigator Main Window**

Project

ISE organizes and tracks your design as a project. A project is a collection of all files necessary to create and download your design to the selected device. The following information is required for each project:

- A unique project name
- A specified target device family (architecture)
- A specified target device
- A specified design flow

Each project has a directory, device family, device, and design flow associated with it as project properties. The project properties enable Project Navigator to display and run only those processes appropriate for the targeted device and design flow.

Sources

A source is any element that contains information about a design. In Project Navigator, you can create and add sources to your project. Each project can contain many sources, each one representing a different part of the overall design. Sources can include the description of circuits (as represented by schematics and hardware description language files), state diagrams, simulation models, test files, and documentation of the design.

Source Hierarchy

One source file in a project is the top-level source for the design. The top-level source defines the inputs and outputs to be mapped into the device, and references the logic descriptions contained in lower-level sources in a hierarchical design. A project must contain at least one source as the top-level source. All source files and their accompanying icons are displayed in the Sources in Project window below the project file.

The term instantiation describes when one source references another. Lower-level sources also can instantiate sources to build as many levels of logic hierarchy as necessary to describe your design.

Valid top-level source types include the following:

- Schematics
- HDL files (VHDL or Verilog)
- EDIF

ISE Tools

ISE includes a number of individual tools and capabilities that can be accessed standalone or within the Project Navigator.

Engineering Capture System (ECS)

The Engineering Capture System (ECS) allows you to create, view, and edit schematics and symbols. You can use ECS to create a top-level schematic and use any of the following to define the lower levels of the design: ECS, CORE Generator System, or HDL code. Then you can translate the schematics created by ECS to a structural HDL for simulation and synthesis, or use the schematics solely for documentation purposes.

HDL Editor

The HDL Editor is a text editor designed especially for editing HDL source files. In addition to regular editing features, the editor provides syntax coloring. The syntax-coloring feature supports both VHDL and Verilog. The HDL Editor operates as a standard text editor as well. ISE provides optimized, ready-to-use language and synthesis templates for easy insertion into an HDL source file.

StateCAD State Machine Editor

StateCAD accelerates design entry by allowing quick entry of finite state machines (FSMs). Each step is automated, reducing development costs and enhancing effectiveness. StateCAD allows users to quickly design FSMs, find and fix design errors, verify behavior, and generate optimized HDL. StateCAD automatically analyzes designs for problems such as stuck-in-states, conflicting state assignments, and indeterminate conditions. This automated error analysis ensures that designs are logically consistent, reducing simulation requirements and improving product reliability.

Xilinx Synthesis Technology (XST)

Xilinx Synthesis Technology (XST) provides cutting edge design optimization techniques from a Xilinx-developed synthesis tool. XST supports the Verilog and VHDL design languages. XST is included in ISE WebPACK, ISE BaseX, and ISE Foundation packages. RTL Viewer displays the results of XST synthesis in a schematic view.

HDL Advisor

The HDL Advisor gives advisory messages in the XST synthesis report files. The messages are designed to make suggestions on how code can be changed to reduce design size and meet timing requirements. These HDL advisors allow designers to produce better code earlier,

reducing design time, and resulting in better space utilization in the Spartan-3 Generation FPGA.

Partner Tools

The Xilinx tools provide easy integration with third-party tools including LeonardoSpectrum synthesis from Mentor Graphics, Synplify/Pro and Amplify synthesis from Synplicity, and FPGA Compiler II from Synopsys. These tools can be purchased separately from the vendor.

ModelSim simulators from Model Technology provide the simulation functions for ISE. ModelSim Xilinx Edition III (MXE-III) is available as an option from Xilinx. It offers a complete PC HDL simulation environment that enables you to verify the HDL source code as well as the functional and timing models of your designs.

Intellectual Property (IP)

Get to market faster and less expensively using the latest pre-verified, pre-optimized Intellectual Property (IP) Cores, Reference Designs, and Design Services for Xilinx FPGAs. Xilinx-created LogiCORE™ products form the most successful core program in the programmable logic industry, including PCI bus interfaces and MicroBlaze™ soft processors. As a result, Xilinx has gained considerable experience developing and selling cores, and servicing FPGA core customers. Through the AllianceCORE™ program, Xilinx is expanding the availability of quality cores for programmable logic by sharing what has been learned with leading third-party core developers. The AllianceCORE program is a cooperative effort between Xilinx and independent third-party core developers. It is designed to produce a broad selection of industry-standard solutions dedicated for use in Xilinx programmable logic. Xilinx also provides many reference designs and design examples provided “as-is” to help get you started with your own designs.

CORE Generator System

The Xilinx CORE Generator System provides a catalog of ready-made functions, ranging in complexity from simple arithmetic operators like adders, accumulators, and multipliers, to system-level building blocks such as filters, transforms, and memory resources. Cores are organized by functional type into folders that expand or contract on demand.

The Xilinx CORE Generator System produces an EDIF netlist, schematic symbol, Verilog template file with a Verilog wrapper file, and a VHDL template file with a VHDL wrapper file. The Electronic Data Netlist (EDN) file contains the information for implementing the module. Cores generated in the Xilinx CORE Generator tool can be used in schematic designs. After the core is selected and customized, the CORE Generator tool generates its schematic symbol. The core then can be added to the schematic like any other library component. Finally, the template files contain code that can be used as a model to instantiate a CORE Generator module in a Verilog or VHDL design so that it can be simulated and integrated into a design.

System Generator for DSP

The System Generator for DSP software enables electronic designs to be created, tested, and translated into hardware for Spartan-3 Generation FPGAs. The tool extends Simulink (from The MathWorks, Inc.) to support bit- and cycle-accurate system-level simulation, and automatic code generation for Xilinx FPGAs. System Generator co-simulation interfaces extend Simulink to incorporate FPGA hardware and HDL simulation into the system-level environment as naturally as other library blocks. System Generator presents a high-level and abstract view of the design, but also exposes key features in the underlying silicon, making it possible to build extremely high-performance FPGA implementations.

Clocking Wizard

To reduce the complexities of new device technologies like Digital Clock Managers (DCM), ISE includes Architecture Wizards, allowing users access through an intuitive easy-to-use dialog. Through the use of the ISE Architecture Wizards, designers can access these leading edge technologies quickly by creating the component through a push-button flow rather than learning

all the attributes in HDL. Then the component simply can be instantiated in the user's design by copying the instantiation template created by ISE. The Clocking Wizard supports all the capabilities of the Spartan-3 Generation DCMs.

Data2BRAM Tool

Data2BRAM is fundamentally a data translation tool. It translates contiguous fragments of data into the proper initialization records for Block RAMs. It automates distribution of that data across multiple physical Block RAMs that constitute a contiguous logical data space. Data2BRAM is also a simplified means for initializing block RAMs.

Automatic Implementation Tools

The automatic implementation tools (synthesis, translation, mapping, placement, and routing) provide the best results for any design. ProActive Timing Closure technologies deliver the industry's highest performance in programmable logic designs, quickly and efficiently. The technologies include:

- Physical Synthesis
  - Includes place and route information to work on the real critical paths first
  - Achieves better quality of results of 5 to 20%
  - Supported through Synplicity's Amplify, Mentor Graphic's LeonardoSpectrum Time Closer, and Xilinx's own XST synthesis tool
  - Timing optimization prior to physical place and route
- Macro Builder
  - Lets you freeze placement information for a given design
  - You can then re-use that macro in future designs using relative placement
  - Performance preservation
- Advanced Place and Route Algorithms
  - Critical Path Placement first
  - Extra-Effort Mode
  - Directed Routing that lets the designer specify routing with IP
- Timing Improvement Wizard
  - Interactively helps designer improve design
  - Click on a timing problem and receive suggestions that can improve design timing
- Timing Cross-Probing
  - Decreases debug time by cross-probing from the timing report directly to Floorplanner
  - Click on the error, path, or net in the timing report and instantly see it in Floorplanner or Synthesis Source Tool
- HDL Advisors
  - Included in XST synthesis reports, clicking on an error or warning suggests changes to HDL to improve the implementation

Incremental Design

Incremental Design gets your overall design to market faster by minimizing the impact from late-arriving design changes. The Incremental Design flow facilitates more debug cycles in a day when making small design changes. A designer quickly and easily can floorplan design areas along hierarchy boundaries, and then finish the design as normal. Later, if a design change is required, Incremental Design ensures that only the area of the design change need be re-implemented; the rest of the design stays locked and intact, delivering overall design completion faster.

Modular Design

Modular Design lets you implement a “divide and conquer” approach to multi-million gate FPGA designs. Partitioning a design into smaller functional modules reduces the complexities of design, implementation, and verification. These design modules then can be brought through the design flow independently, leveraging all of the powerful tools within the Xilinx FPGA design flow. Once completed, a module's implementation is preserved, guaranteeing the timing in the finished device. This technology is a requirement for any organization employing a team design methodology for the design of a multi-million gate FPGA.

Constraints Editor

Constraints are user instructions placed on elements of a schematic or HDL design, either in the design itself or in a separate file. They can indicate a number of things such as placement, implementation, naming, signal direction, and timing considerations. In the Xilinx development system, logical constraints are placed in a file called the UCF (User Constraints File). The Constraints Editor is a graphical program that you can use to create and modify those constraints.

PACE

Pinout and Area Constraints Editor (PACE) is an interactive graphical application that you can use to do the following functions:

- View and edit location constraints for I/Os and global logic
- View and create area constraints for hierarchical symbols in your design
- Determine connectivity and resource requirements of your design
- Determine resource layout of your target FPGA
- Determine how your design maps onto the FPGA via location and area constraints

PACE fits into the Xilinx implementation flow at the very beginning. Because PACE supports I/O layout with an NGD file, it can be used early at the design entry stage of the flow. PACE reads an NGD file and reads and writes a UCF file.

Floorplanner

Use the Floorplanner interactive graphical tool to perform the following functions on your designs:

- Floorplan resource placement at a detailed level
- Use Macro Builder to create a Relationally Placed Macro (RPM) core that can be used in other designs
- View and edit location constraints
- Find logic or nets by name or connectivity
- Cross-probe from the Timing Analyzer to the Floorplanner
- Automatic placement of ports for modular design

The graphical user interface includes pull-down menus and toolbar buttons that contain all of the necessary commands for changing the design hierarchy, floorplanning, and performing design rule checks. Dialog boxes allow you to quickly set parameters and options for command execution.

FPGA Editor

The FPGA Editor is a graphical application for displaying and configuring FPGAs. The FPGA Editor requires an NCD file. This file contains the logic of your design mapped to components such as CLBs and IOBs. In addition, the FPGA Editor reads from and writes to a Physical Constraints File (PCF).

The following is a list of a few of the functions you can perform on your designs in the FPGA Editor:

- Place and route critical components before running automatic place and route
- Fine-tune placement and routing after running automatic place and route
- Add probes to design to examine the signal states of the targeted device
- Run the Bitstream Generator and download the resulting file to the targeted device
- View and change the nets connected to the capture units of an Integrated Logic Analyzer (ILA) core
- Create an entire design by hand (for advanced users)

HDL Bencher Software

The HDL Bencher software automates verification of VHDL sources, Verilog sources, and schematics created within ISE. Design sources are imported, a waveform is created, and stimulus is specified by filling in the WaveTable spreadsheet cells. Outputs may be auto-simulated via a command from ISE. A self-checking test bench is exported whenever the waveform is saved. No knowledge of HDL or language scripting is needed to verify the design functions as intended.

Multiple layers of simulation are supported. Waveforms that include the expected timing results are developed for behavioral designs. The waveforms may be simulated behaviorally, after translation, after mapping, or after routing.

The HDL Bencher software constrains the test run to a specific sequence of events, initial conditions, and user-determined results. With the HDL Bencher software, you quickly can validate your design functions as intended.

Interactive Timing Analyzer

The Interactive Timing Analyzer provides a powerful, flexible, and easy way to perform static timing analysis. With Timing Analyzer, analysis can be performed immediately after mapping, placing, or routing a Spartan-3 Generation FPGA design.

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

Timing Analyzer creates timing analysis reports based on existing timing constraints or user specified paths within the program. Timing reports have a hierarchical browser to quickly jump to different sections of the reports. Timing paths in reports can be cross-probed to synthesis tools (Exemplar and Synplicity) and the Floorplanner.

iMPACT Configuration Tool

The iMPACT configuration tool, a command line and GUI based tool, allows you to configure your PLD designs using Boundary Scan, Slave Serial, SelectMap, and Desktop Configuration modes. It also allows you to do the following:

- Download
- Read back and verify design configuration data
- Debug configuration problems
- Create PROM, SVF, STAPL, System ACE™ CF, and System ACE MPM programming files

ChipScope Pro Analyzer

The ChipScope Pro analyzer delivers in-circuit real-time debugging with shorter verification cycles and lower project costs. By inserting special low-impact IP debugging cores directly into

your HDL code or design netlist, you can debug and verify FPGA logic and system bus activity, capturing signals at or near system operating speeds. You easily can change your trace points without having to recompile your design. The ChipScope Pro analyzer embeds Integrated Logic Analyzer (ILA) and Integrated Bus Analyzer (IBA) cores into your design. These cores allow the user to view all the internal signals and nodes within the Spartan-3 Generation FPGA.

XPower Analysis Tool

XPower is a post-route analysis tool for interactively and automatically analyzing power consumption for Xilinx devices. XPower includes both GUI (XPower) and batch (xpwr) applications.

Earlier in the design flow than ever, you can analyze total device power, power per net, routed, partially routed or unrouted designs, all driven from a comprehensive graphical interface or command-line driven batch mode. XPower also reads VCD simulation data from the ModelSim family of HDL simulators to set estimation stimulus, reducing setup time, as well as from additional simulators.

XPower uses device knowledge and design data to estimate device power and by-net power utilization. Information is presented in both HTML and ASCII (text) report formats. The accuracy of XPower is higher than the power estimator worksheets available for pre-design analysis.

Related Materials and References

The following documents provide supplementary information useful with this application note:

- Xilinx Design Tools Center
  http://www.xilinx.com/products/design_resources/design_tool/
- ISE 7.1i for Spartan-3 Generation Designs
  http://www.xilinx.com/ise/ise_promo/ise_spartan3.htm
- Software Download Center
  http://www.xilinx.com/xlnx/xil_sw_updates_home.jsp
- Software Manuals
  http://www.xilinx.com/support/software_manuals.htm

Conclusion

The ISE design environment brings you the fastest, most complete family of design tools available. The ISE tools are available in multiple configurations with various optional tools and interfaces to third-party tools, allowing you to customize the set of tools for your own needs. ISE combines advanced technologies such as ProActive Timing Closure with a flexible, easy-to-use graphical interface to help you achieve the best possible designs with the least time and effort, regardless of your experience level.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
07/11/03	1.0	Initial Xilinx release.
05/23/05	1.1	Included all Spartan-3 Generation families, including Spartan-3E and Spartan-3L devices. Made additional clarifications.