Development System Reference Guide

Development System Reference Guide — 3.1i *Printed in U.S.A.*

SPEEDPRINT

BitGen

PROMGen

NGDAnno

NGD2EDIF

NGD2VER

NGD2VHDL

XFLOW

Xilinx Development System Files

EDIF2NGD, XNF2NGD, and NGDBuild

About This Manual

The *Development System Reference Guide* contains information on the software programs in the Xilinx Development System. Generally, the chapters are organized in the following way.

- A brief summary of program functions
- A syntax statement
- A review of the input files used and the output files generated by the program
- A listing of the commands, options, or parameters used by the program
- Examples of how you can use the program

For an overview of the Xilinx Development System describing how these programs are used in the design flow, see the "Design Flow" chapter.

Note This Xilinx software release is certified as Year 2000 compliant

Manual Contents

The *Development System Reference Guide* provides detailed information about converting, implementing, and verifying designs in the Xilinx environment. Check the program chapters for information on what program works with each family of Field Programmable Gate Array (FPGA) or Complex Programmable Logic Device (CPLD). The following is a brief overview of the contents and organization of the *Development System Reference Guide*.

- Chapter 1, “Introduction” —This chapter describes some basics that are common to the different Xilinx Development System modules.
- Chapter 2, “Design Flow”—This chapter describes the basic design processes: design entry, implementation, and verification.
- Chapter 3, “PARTGEN”—The PARTGEN command allows you to obtain information about installed devices and families
- Chapter 4, “NGDBuild”—NGDBuild performs all of the steps necessary to read a netlist file in XNF or EDIF format and create an NGD (Native Generic Database) file describing the logical design reduced to Xilinx primitives.
- Chapter 5, “User Constraints (UCF) File”—The UCF File is an ASCII file in which you enter constraints affecting how the logical design is implemented.
- Chapter 6, “Using Timing Constraints”—This chapter describes how you specify timing requirements for your design.
- Chapter 7, “Logical Design Rule Check”—The Logical DRC (Design Rule Check) is a series of tests run to verify the logical design described by the NGD (Native Generic Database) file.
- Chapter 8, “MAP—The Technology Mapper”—MAP maps the logic defined by an NGD file into FPGA elements such as CLBs, IOBs, and TBUFs.
- Chapter 9, “LCA2NCD”—LCA2NCD translates an LCA file from an earlier Xilinx Development System release to an NCD file.
- Chapter 10, “Physical Constraints (PCF) File”—The PCF file is an ASCII file containing physical constraints created by the MAP program and physical constraints you enter.
- Chapter 11, “DRC—Physical Design Rule Check”—The physical Design Rule Check (DRC) consists of a series of tests used to discover physical errors in your design.
- Chapter 12, “PAR—Place and Route”—PAR places and routes FPGA designs.
- Chapter 13, “PIN2UCF”—PIN2UCF generates pin locking constraints in a UCF file by reading a a placed NCD file for FPGAs or GYD file for CPLDs.

- Chapter 14, “TRACE”—TRACE (Timing Reporter and Circuit Evaluator) performs static timing analysis of the physical design based on input timing constraints.
- Chapter 15, “SPEEDPRINT”— SPEEDPRINT lists block delays for a specified device and its speed grades.
- Chapter 16, “BitGen”—BitGen creates a configuration bitstream for an FPGA design.
- Chapter 17, “PROMGen” —PROMGen converts a configuration bitstream (BIT) file into a file that can be downloaded to a PROM. PROMGen also combines multiple BIT files for use in a daisy chain of FPGA devices.
- Chapter 18, “NGDAnno”—NGDAnno annotates timing information found in the physical NCD design file back to the logical NGD file.
- Chapter 19, “NGD2EDIF”—NGD2EDIF converts an NGD file to an EDIF file for use in simulation.
- Chapter 20, “NGD2VER”—NGD2VER converts an NGD file to a Verilog HDL file for use in simulation.
- Chapter 21, “NGD2VHDL”—NGD2VHDL converts an NGD file to a VHDL file for use in simulation.
- Chapter 22 “XFLOW”—XFLOW is a command tool that runs the full suite of implementation and simulation flows.
- Appendix A, “Xilinx Development System Files”—This appendix gives an alphabetic listing of the files used by the Xilinx Development System.
- Appendix B, “EDIF2NGD, XNF2NGD, and NGDBuild” —This appendix describes the netlist readers (EDIF2NGD and XNF2NGD) and how they interact with NGDBuild.

Additional Resources

For additional information, go to `http://support.xilinx.com`. The following table lists some of the resources you can access from

this Web site. You can also directly access these resources using the provided URLs.

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

Conventions

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

Typographical

The following conventions are used for all documents.

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

  ```
  speed grade: - 100
  ```

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

  `rpt_del_net=`

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

  `File` → **`Open`**

- *Italic font* denotes the following items.

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

    `edif2ngd` *`design_name`*

  - References to other manuals

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

- Emphasis in text

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

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

  edif2ngd *[option_name] design_name*

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

  lowpwr ={on|off}

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

  lowpwr ={on|off}

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

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

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

  `allow block` *block_name loc1 loc2locn;*

Online Document

The following conventions are used for online documents.

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

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

Contents

Chapter 3 PARTGEN

Chapter 4 NGDBuild

Chapter 5 User Constraints (UCF) File

Chapter 6 Using Timing Constraints

Chapter 12 PAR—Place and Route

Chapter 13 PIN2UCF

Chapter 14 TRACE

Chapter 15 SPEEDPRINT

Chapter 16 BitGen

Chapter 17 PROMGen

Chapter 20 NGD2VER

Chapter 21 NGD2VHDL

Chapter 22 XFLOW

Appendix A Xilinx Development System Files

Appendix B EDIF2NGD, XNF2NGD, and NGDBuild

Chapter 1

Introduction

This chapter describes some basics that are common to the different Xilinx Development System modules. The chapter contains the following sections.

- "Invoking Xilinx Development System Programs"
- "Command Line"
- "Reading NCD Files with NCDRead"
- "Terminology"
- "Supported Platforms"

Invoking Xilinx Development System Programs

You can start Xilinx Development System programs in the following ways.

- Enter a command at the UNIX™ command line or on a DOS command line in an MS-DOS™ Prompt window (Windows 95®) or a Command Prompt window (Windows NT®).
- Invoke a command from one of the following Xilinx graphical applications.
  - Design Manager / Flow Engine
  - Foundation Project Manager
  - Timing Analyzer
  - FPGA Editor
  - Hardware Debugger
  - PROM File Formatter

Note The graphical applications are described in separate manuals. This reference manual describes only the command line interface.

Command Line

Command line options are entered on the command line in any order, preceded by a hyphen (–), sometimes preceded by a + (plus sign), and separated by spaces. Most command line options are case-sensitive. When an option requires an additional parameter, that parameter must be separated from the option letter by spaces or tabs (for example, **-l 5** is correct, **-l5** is not).

Files are position-dependent. For example, **par input.ncd output.ncd freq.pcf** is legal; **par input.ncd freq.pcf output.ncd** is not. File extension use is case-sensitive. All file extensions (for example, .ncd) must be in lower case for all command line tools.

For options that can be specified multiple times, the option letter must, in most cases, precede each parameter. For example, **–l xilinxun synopsys** is not acceptable, while **–l xilinxun -l synopsys** is allowed.

Options can appear anywhere on the command line. Arguments that are bound to a particular option must appear after the option. For example, **–f** *command_file* is legal; *command_file* **–f** is not.

When you enter the Xilinx Development System application name on the command line with no arguments and the application requires one or more arguments (PAR, for example), a message appears consisting of the command line format string. The format string contains the following symbols, along with literals

Symbol	Description
[]	Encloses items that are optional
{}	Encloses items that may be repeated zero or more times.
<>	Encloses a variable name or number for which you must substitute information.
, (comma)	Indicates a range for an integer variable.
– (dash)	Indicates the start of an option name.
+	Indicates the start of an option name.

Symbol	Description
:	The bind operator. Binds a variable name to a range.
|	Logical OR to indicate a choice of one out of many items. The OR operator may only separate logical groups or literal keywords.
()	Encloses a logical grouping for a choice between subformats.

When you enter the Xilinx Development System application name on the command line followed by **–help** or **–h**, a message displays that explains each of the options and arguments. For example, when you type **edif2ngd –h**, the following message appears.

```
edif2ngd:  version 3.1i
Copyright (c) 1995-2000 Xilinx, Inc.  All rights reserved.
Usage: edif2ngd [-a] [-r] {-l <library>} [-p <partname>] <edif_file>
[<ngo_file>]
      -a            Add PAD's to all top level port signals
      -r            Remove LOC props from the design
      -l library    Design is built from <library>
      -p partname   Override/define part name in EDIF file
      <edif_file>   EDIF 2 0 0 format file
      <ngo_file>    Output '.ngo' file. Default is <infile>.ngo.
```

To redirect this message to a file (to read later or to print out), enter the following.

command_name **–help >&** *filename*

For Xilinx Development System applications that have architecture-specific command lines, enter the application name plus –help (or –h) plus the architecture to get the verbose message specific to that architecture. If you do not specify the architecture, a message similar to the following appears.

```
Use '<appname> -help <architecture>' to get
   detailed
usage for a particular architecture.
```

Notes about Screen Messages

<*infile*[**.ncd**]> indicates that the .ncd extension is optional but that the extension must be .ncd.

<*infile*<**.xnf**>> indicates that the .xnf extension is optional and is appended only if there is no other extension in the file name.

Part Numbers in Commands

The EDIF2NGD, XNF2NGD, NGDBuild, MAP, and XFLOW commands have options to specify the part into which your design will be implemented. A complete Xilinx part number consists of these elements.

- Architecture (for example, xc4000ex)
- Device (for example, xc4028ex)
- Package (for example, pq208)
- Speed (for example, -3)

The following table shows the various way to specify a part on the command line.

Specification	Examples
Architecture only	4000ex x4000ex xc4000ex
Device only	4028ex x4028ex xc4028ex
DevicePackage	4028exhq240 x4028exhq240 xc4028exhq240
Device–Package	4028ex-hq240 x4028ex-hq240 xc4028ex-hq240
DevicePackage–Speed	4028exhq240-3 x4028exhq240-3 xc4028exhq240-3

Specification	Examples
Device–Package–Speed	4028ex-3-hq240 x4028ex-3-hq240 xc4028ex-3-hq240
Device–Speed	4028ex-3-hq240 x4028ex-3-hq240 xc4028ex-3-hq240
Device–Speed–Package	4028ex-3-hq240 x4028ex-3-hq240 xc4028ex-3-hq240
Device–SpeedPackage	4028ex-3hq240 x4028ex-3hq240 xc4028ex-3hq240

Note SPEEDPRINT requires a device name (part number) that is somewhat different from the previous table. See the "SPEEDPRINT" chapter for details.

You can specify a part number at a number of points in the design flow. A part number specified in a later step of the design flow overrides a part number specified in an earlier step.

The following list below indicates the points in the design flow when you can specify a part number. In the list, a part number specified later in the design process overrides a specification at an earlier level. As an example, a part specified when you run MAP (the last bulleted item) overrides a part specified at any other step in the design flow.

- There may be a part specified in the input netlist.
- There may be a part specified in an NCF (Netlist Constraints File).
- You can specify a part with the –p option when you run a netlist reader (EDIF2NGD or XNF2NGD).
- You can specify a part in a UCF (User Constraints File).
- You can specify a part with the –p option when you run NGDBuild.

  When you run NGDBuild, you must have already specified at least a device architecture
- You can specify a part with the –p option when you run MAP.

When you run MAP, an architecture, device, and package must be specified, either on the MAP command line or earlier in the design flow. MAP selects a default speed if none has been specified. You can only run MAP for a part from the architecture you specified when you ran NGDBuild.

–f Option

For any Xilinx Development System executable, you can store arguments (that is, file names and command options) in a file and then execute the arguments at any time by entering the –f option on the UNIX or DOS command line followed by the name of the file containing the arguments. This can be useful if you frequently execute the same arguments each time you perform the command, or if the command line becomes too long.

You can use the options file in the following two ways.

- To supply all the command arguments, as in this example.

  par –f *command_file*

  command_file is the name of the file containing the command-line arguments.

- To insert certain command line arguments within the command line, as in this example.

  ```
  par –i 33 –f placeoptions –s 4 –f routeoptions
  design_i.ncd design_o.ncd
  ```

  placeoptions is the name of a file containing placement command arguments.

  routeoptions is the name of a file containing routing command arguments.

The space between the –f flag and the file name is required.

The command file is an ASCII file containing the command arguments. Arguments are separated by a space and can be spread across one or more lines within the file. You can put new lines or tabs anywhere white space is allowed on the UNIX or DOS command line. You can put all arguments on the same line, or one argument per line, or a combination of these. There is no line length limitation within the file.

All carriage returns and other non-printable characters are treated as space and ignored. Comments are designated by a # (pound sign) and go to the end of the line.

Following is an example of a command file.

Sample Command File

```
#command line options for par for design mine.ncd
-a  -n 10
-w
-l 5
-s 2 #will save the two best results
/home/users/jimbob/designs/xilinx/mine.ncd
#directory for output designs
/home/users/jimbob/designs/xilinx/output.dir
#use timing constraints file
/home/users/jimbob/designs/xilinx/mine.pcf
```

Reading NCD Files with NCDRead

An NCD (Native Circuit Description) file contains a physical description of your design in terms of the components in the target architecture. NCDRead enables you to quickly generate an ASCII (text) file based on the data found in one or more NCD files.

To start NCDRead from the UNIX or DOS command line, type the following.

ncdread [**–o** *outfile_name*] *filename0*.**ncd** {*filename1*.**ncd** ...}

Note Standard output goes only to your screen if you do not use the –o option to write the output to a file.

Following is an example of an output file from NCDRead. The example gives information on three of the components in the design. An actual file includes information on all the components.

```
Loading device database for application ncread from file
"testclk.ncd"."testclk" is an NCD, version 2.28, device xcv100,
package bg256, speed -4

Loading device for application ncread from file 'v100.nph' in
environment
/build/bcxfndry/C.13/rtf.
```

```
NC_DESIGN <testclk> - version 2.28
  vendor = Xilinx, package = bg256, speed = -4
  72 comps

NC_BEL:5 - <C253> ngdid = 5
cmprim = <lut-or-mem>
comp = <core_inst/counter2/cont<3>>, cmid = 48
signals on pin:
  A1 - core_inst/counter2/C54/N32
  A2 - core_inst/counter2/N356
  A3 - syn2748
  D - core_inst/counter2/C65/N19
  bel config info:
      Inverted pins:
      clkedge=CM_CLKRISING
      O=(~I0*I1*~I2)+(~I0*I1*I2)+(I0*I1*I2)
      lutmode is EQN_MODE_LUT
      clk_is_inverted=FALSE,shift_register=FALSE
.
.
.
NC_BEL:7 - <core_inst/counter2/cont_reg<3>> ngdid = 5
cmprim = <latch-or-ff>
comp = <core_inst/counter2/cont<3>>, cmid = 73
signals on pin:
  CK - ck2
  D - core_inst/counter2/C65/N19
  Q - core_inst/counter2/cont<3>
  INIT - core_inst/counter1/n224
  bel config info :
      Inverted pins:
      clkedge=CM_CLKRISING
      clkinvert=FALSE,resetmode=TRUE resetedge=CM_SRFALLING
      resetinvert=FALSE
.
.
.
NC_COMP:3 - <core_inst/counter2/cont<3>> ngid = 5, external = 0
siteparam:10, site:-1, l2pmap:-1, macro:-1
      Config String: <CYSELF:#OFF CYSELG:#OFF CKINV:1 COUTUSED:#OFF
      YUSED:#OFF XUSED:#OFF XBUSED:#OFF F5USED:#OFF YBMUX:#OFF
```

```
      CYINIT:#OFF DYMUX:1 DXMUX:1 CY0F:#OFF CY0G:#OFF
      F:#LUT:D=(~A1*A2*~A3)+(~A1*A2*A3)+(A1*A2*A3)
      G:#LUT:D=(~A1*A2*~A3)+(~A1*A2*A3)+(A1*A2*A3) RAMCONFIG:#OFF
      REVUSED:#OFF BYMUX:#OFF BXMUX:#OFF CEMUX:#OFF SRMUX:SR
      GYMUX:GFXMUX:F  SYNC_ATTR:ASYNC  SRFFMUX:0  INITY:LOW  FFX:#FF
FFY:#FF
      INITX:LOW>
      There are <10> paths.

  17:  [f:12]  8  pins  F1:F1  F:A1  F:D  FXMUX:F  FXMUX:OUT  DXMUX:1
DXMUX:OUT FFX:D Container (1): 18
  43:  [f:12]  8  pins  F2:F2  F:A2  F:D  FXMUX:F  FXMUX:OUT  DXMUX:1
DXMUX:OUT FFX:D
Container (1): 44
  69:  [f:12]  8  pins  F3:F3  F:A3  F:D  FXMUX:F  FXMUX:OUT  DXMUX:1
DXMUX:OUT FFX:
Container (1): 70
  89:  [f:12]  8  pins  G1:G1  G:A1  G:D  GYMUX:G  GYMUX:OUT  DYMUX:1
DYMUX:OUT FFY:D
Container (1): 90
  107:  [f:12]  8  pins  G2:G2  G:A2  G:D  GYMUX:G  GYMUX:OUT  DYMUX:1
DYMUX:OUT FFY:D

Container (1): 108
  123:  [f:12]  8  pins  G3:G3  G:A3  G:D  GYMUX:G  GYMUX:OUT  DYMUX:1
DYMUX:OUT FFY:D

Container (1): 124
  531: [f:8] 6 pins CLK:CLK CKINV:1 CKINV:OUT FFX:CK FFX:Q XQ:XQ
  532: [f:8] 6 pins CLK:CLK CKINV:1 CKINV:OUT FFY:CK FFY:Q YQ:YQ
  569: [f:8] 8 pins SR:SR SRMUX:SR SRMUX:OUT SRFFMUX:0 SRFFMUX:OUT
FFX:INIT

FFX:Q XQ:XQ
  570: [f:8] 8 pins SR:SR SRMUX:SR SRMUX:OUT SRFFMUX:0 SRFFMUX:OUT
FFY:INIT

FFY:Q YQ:YQ

            There are <26> delays.
            Pin Types:
                Real_pin=<00> pintype=<0x10,16>
```

```
Real_pin=<00> pintype=<0x10,16>
.
.
.
```

Terminology

Commonly used terms in the Xilinx Development System are defined in this section. Terms specific to certain Xilinx Development System modules are described in the relevant chapters.

- A *device* is a particular FPGA or CPLD. For example, a Xilinx XC4010E is a device.
- A *site* is a programmable logic element (used or unused) located within the device.
- A *component* is a logic configuration that will, at some point, go into a physical site. Examples of components are CLBs, IOBs, tristate buffers, pull-up resistors, and oscillators.
- A *net* (also called a signal) is a set of two or more component pins to be electrically connected in the finished design. A net normally consists of a driver pin and one or more load pins, but it may have more than one driver pin in certain cases. A net does not pass through a logic block, except in the case of route-throughs (routes that pass through occupied or unoccupied logic sites). The following figure shows two examples of nets. In the example, Net 1 consists of a driver pin (A) and a single load pin (B). Net 2 consists of a driver pin (A) and multiple load pins (B, C, and D). The net contains a route-through at component COMP_1.

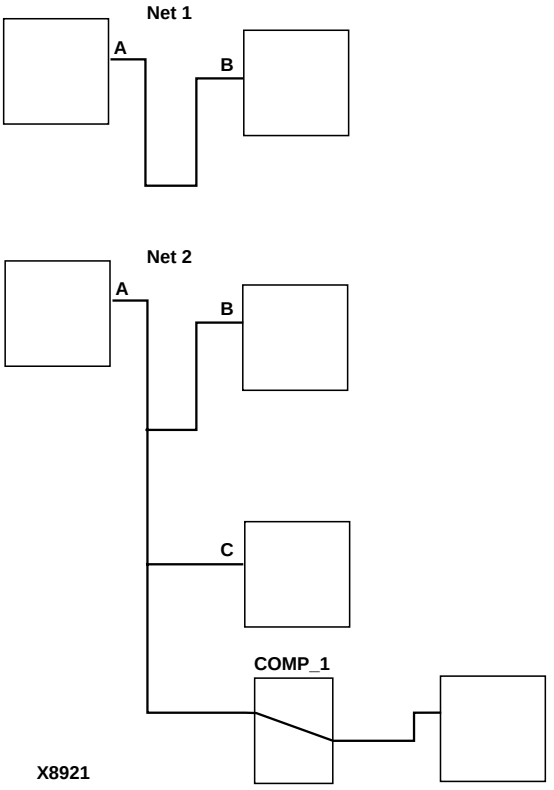


Figure 1-1 Net Example

- A *path* is an ordered set of elements identifying a logic flow pathway through a circuit. A path can consist of a single net or a grouping of related nets and components. You can have multiple paths (consisting of nets and components) between the two pins. When a component is selected as part of a path, both the input pin to the component and the output pin are included in the path.

  A path starts by including a clock-to-out delay at a synchronous element (flip flop, RAM, or latch) or from a pad. The path continues adding net delays and combinatorial delays. A path ends with a setup-to-clock delay at an asynchronous element (flip

flop, RAM, or latch) or at a pad. The following figure shows a path through CLB1, CLB2, and CLB3.

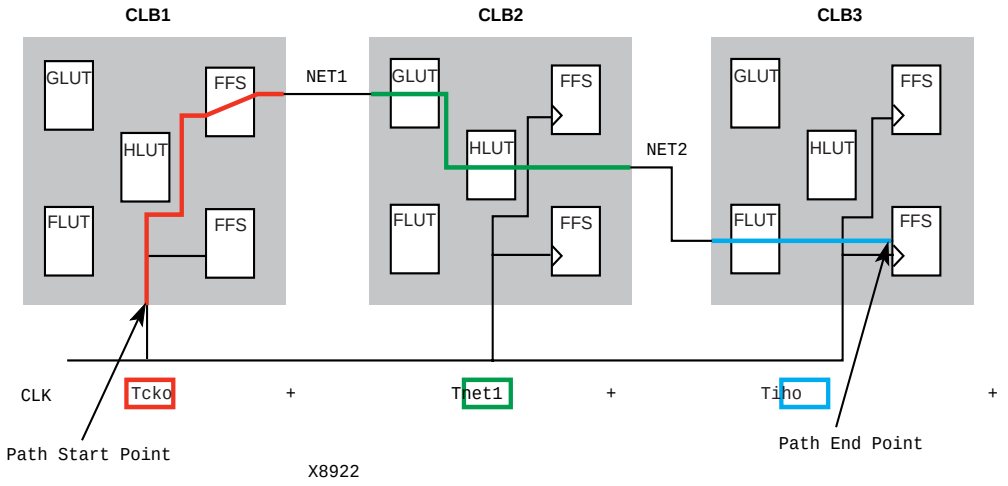


Figure 1-2 Path Example

- A *bus* is a grouping of related nets. For example, you can create a bus containing the nets DATA_00, DATA_01, DATA_02 and DATA_03—nets that supply data to RAM.
- A *BEL* is a Basic ELement. BELs are the building blocks that make up a component (CLB or IOB)—function generators, flip-flops, carry logic, and RAMs.
- A *physical macro* is a logical function such as a counter that is created from a set of physical components for a specific device family. Physical macros, which are created using the FPGA Editor, are stored in files with the .nmc extension. In addition to components and nets, the file can also contain relative placement and / or routing information. A physical macro can be unplaced, partially placed, or fully placed, and it can also be unrouted, partially routed, or fully routed. Note that you cannot perform functional simulation of a physical macro. See the “Working with Physical Macros” chapter of the *FPGA Editor Guide* for information about physical macros.

Supported Platforms

The Alliance 3.1i software supports several different operating systems, as shown in the following table.

Platform Type	Version Number
Solaris	Solaris 2.6 and 2.7
HP Series 9000	HP-UX 10.20 and 11.0
Windows NT	NT 4.0, 98 / 2000

Chapter 2

Design Flow

This chapter describes the process for creating, implementing, verifying, and downloading designs for FPGA and CPLD devices. For a complete description of FPGAs and CPLDs, refer to *The Programmable Logic Data Book*.

This chapter contains the following sections.

- "Overview"
- "Design Entry"
- "Design Implementation"
- "FPGA Editor"
- "FPGA Design Techniques"
- "Design Verification"

Overview

The design flow is a three-step process that consists of the following stages

- Design Entry — In this stage 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.
- Design Implementation — By implementing to a specific Xilinx architecture, you convert the logical design file format, such as EDIF, that you created in the design entry stage into a physical file format. The physical information is contained in the NCD (Native Circuit Description) file for FPGAs and the VM6 file for CPLDs. Then you create a bitstream file from these files and optionally program a PROM or EPROM for subsequent programming of your Xilinx device.

- Design Verification — Using a gate level simulator, the Xilinx XChecker™ cable, the Parallel Cable III, or MultiLINX™ cable, you ensure that your design meets your timing requirements and functions properly. See the *Hardware User Guide* for more information about Xilinx download cables and demonstration boards.

The Xilinx design flow is shown in the following figure.

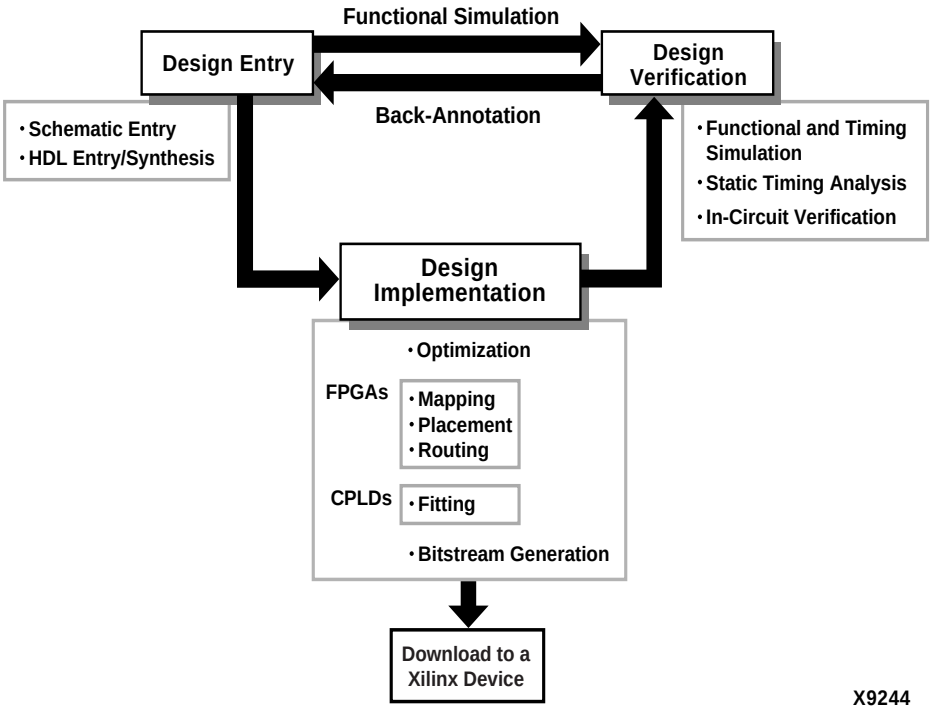


Figure 2-1 Xilinx Design Flow Overview

The full design flow is an iterative process of entering, implementing, and verifying your design until it is correct and complete. The Xilinx Development System allows quick design iterations through the design flow cycle. Because Xilinx devices permit unlimited reprogramming, you do not need to discard devices when debugging your design in-circuit.

The following table defines the terms used in the preceding figure.

Table 2-1 Design Flow Terms

Term	Description
Schematic entry	Creation of designs using graphic symbols
HDL Entry / Synthesis	Creation of designs using Hardware Description Language (HDL) or state machine editor
Optimization	Conversion of device-independent or behavioral logic descriptions to a form that can be efficiently implemented in a Xilinx device
Mapping	Representation of a design's logic as resources of the Xilinx FPGA
Placement	Assignment of design blocks created during mapping to specific locations in the FPGA
Routing	Assignment of the interconnect paths in FPGAs
Fitting	Assignment of logic from your design into physical macrocell locations in the CPLD. Routing is performed automatically, and because of the UIM architecture, all designs are routable.
Bitstream generation	Conversion of a design into a bitstream that can be loaded into a Xilinx device
Back-annotation	Association of implementation net delay information with the original nets found in the input design
Simulation	Emulation of a design‘s logic and timing using input stimuli

The following figure shows the Xilinx software flow chart for designs.

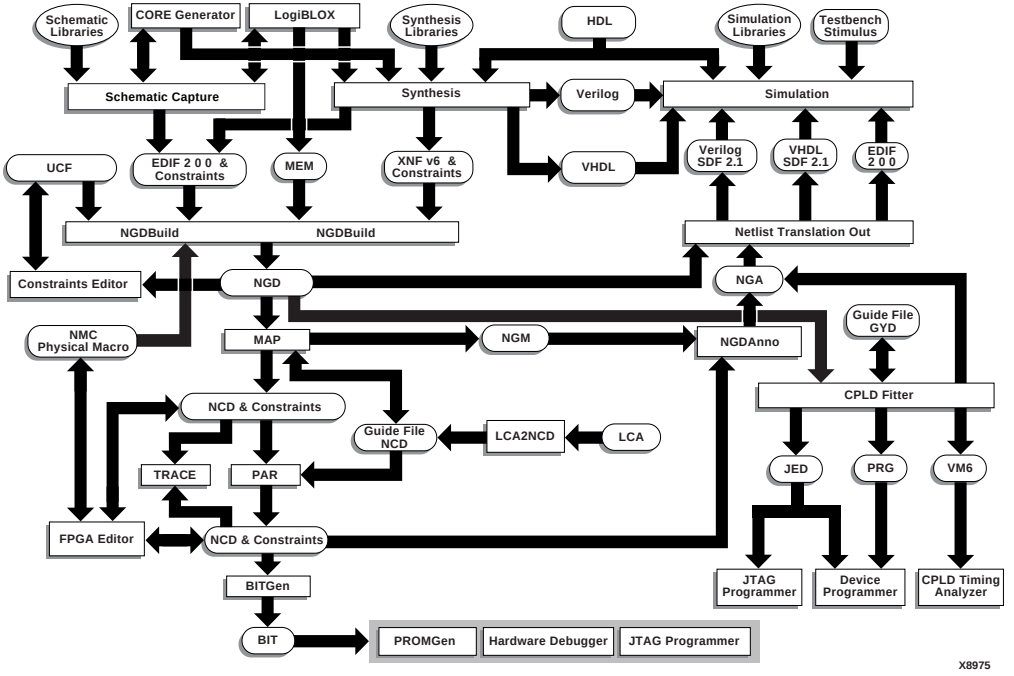


Figure 2-2 Xilinx Software Design Flow

Design Entry

This section introduces the Xilinx design entry process. You can enter a design with a schematic editor or a text-based tool. For the Alliance software, these entry methods require Xilinx-supported third-party tools, which produce a design file in some third party netlist formats. For Foundation, you access the design entry tools from the Foundation Project Manager.

Design entry begins with a design concept, expressed as a drawing or functional description. From the original design, a netlist is created, then synthesized and translated into a Native Generic Object (NGO) file. This file is fed into a program called NGDBuild, which produces a logical Native Generic Database (NGD) file.

The following figure illustrates the design entry process.

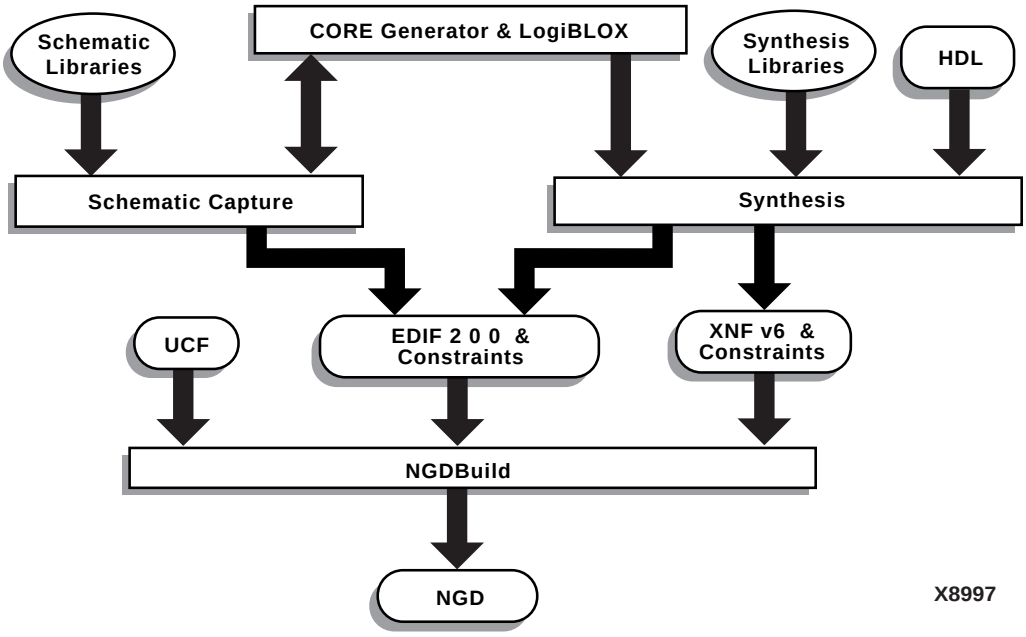


Figure 2-3 Design Entry Flow

The following sections describe schematic and text-based HDL design entry methods in detail.

Schematic Entry Overview

Schematic tools provide a graphic 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. This section focuses on ways to enter functional blocks using library elements and the LogiBLOX and CORE Generator tools.

Library Elements

The following section discusses primitives and macros, which are the "building blocks" of component libraries.

Xilinx libraries provide primitives as well as common high-level macro functions. Primitives are basic circuit elements, such as AND

and OR gates. Each primitive has a unique library name, symbol, and description. Macros contain multiple library elements, which can include primitives and other macros.

There are two types of macros you can use with Xilinx FPGAs. Soft macros, available for all FPGAs, have pre-defined functionality, but have flexible mapping, placement, and routing. Relationally placed macros (RPMs) have fixed mapping and relative placement. They are are available for all device families except the XC3000, XC3100, and XC9500 families.

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 LogiBLOX or CORE Generator modules. For most leading-edge synthesis tools, this does not offer an advantage unless it is for a module that cannot be inferred.

Hierarchical Design

Schematics usually contain hierarchy, which is important 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; incremental design consists of designing, implementing, and verifying individual sub-blocks of a design in stages
- Reduces optimization time
- Facilitates concurrent design; concurrent design is the process of dividing a design among a number of people who develop different parts of the design in parallel.

Hierarchical Names

A specific hierarchical name identifies each library element, unique block, and instance you create. The following example shows a hierarchical name with a 2-input OR gate in the first instance of a multiplexer in a 4-bit counter.

```
/Acc/alu_1/mult_4/8count_3/4bit_0/mux_1/or2
```

Note Xilinx strongly recommends that you name the components and nets in your design. These names are preserved and used by the FPGA Editor. 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 schematic editor automatically generates the names. For example, if left unnamed, the software might name the previous example as follows.

```
/$1a123/$1b942/$1c23/$1d235/$1e121/$1g123/$1h57
```

It can be very difficult to analyze circuits with automatically generated names, because they only have meaning for Xilinx software.

CORE Generator Tool (FPGAs Only)

The Xilinx CORE Generator design tool delivers parameterizable cores that are optimized for Xilinx FPGAs. The library includes cores ranging from simple delay elements to complex DSP (Digital Signal Processing) filters and multiplexers. For details, refer to the *CORE Generator Guide*. You can also refer to the Xilinx IP (Intellectual Property) Center Web site at `http://www.xilinx.com/ipcenter`, which offers the latest IP solutions. These solutions include design reuse tools, free reference designs, DSP and PCI solutions, IP implementation tools, cores, specialized system level services, and vertical application IP solutions.

LogiBLOX Tool

The LogiBLOX tool can generate a variety of variable-sized MSI- and LSI-level design building blocks such as adders, counters, decoders, and shift registers. These modules complement the Xilinx macro libraries, which contain simpler, fixed-size logic and gate functions. The LogiBLOX tool also integrates these modules into your design. For further information, see the *LogiBLOX Guide*.

Note The LogiBLOX tool does not support the Virtex, Virtex-II, Virtex-E, and Spartan-II FPGA families. For these families, use the CORE Generator tool.

HDL Entry/Synthesis

Hardware Description Languages (HDLs) and their associated simulators and synthesizers are powerful tools for integrated circuit designers. A typical 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 very high level of abstraction, then incrementally refine a design's detailed gate-level implementation.

HDL descriptions play an important role in modern design methodology for the following reasons.

- 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 higher level, before implementation at the gate-level, 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.

Xilinx supports HDL synthesis tools for several third-party synthesis vendor partners.

Controlling Implementation with Constraints

Before you implement your design, you may want to constrain it within certain timing or placement parameters. You can specify mapping, block placement, and timing specifications during design entry. The following sections describe these methods.

Mapping (FPGAs Only)

You can specify how a particular block of logic is mapped into CLBs using a CLBMAP for all XC3000 and XC3100 FPGA families; an FMAP or HMAP for all XC4000 and Spartan FPGA families; an FMAP for all Virtex FPGA families; or an FMAP or F5MAP for XC5200 FPGAs. These mapping symbols can be used in your schematic. However, if you overuse these specifications, it may be harder to route your design.

Block Placement

Block placement can be constrained to a specific location, to one of multiple locations, or to a location range. Locations can be specified in the schematic, with synthesis tools, or in the User Constraint File (UCF). Poor block placement can adversely affect both the placement and the routing of a design. Typically, block placement defines I/O placement. For details about placement constraints, refer to the "Placement Constraints" section in the *Libraries Guide*.

Timing Specifications

You can specify timing requirements for paths in your design directly in your schematic. PAR (the Xilinx FPGA Place and Route program) uses these timing specifications to achieve optimum performance when placing and routing your design. See the *Timing Analyzer Guide* and the *Constraints Editor Guide* for a detailed explanation of timing specifications. Also refer to the "Using Timing Constraints" chapter in this manual, the *Development System Reference Guide.*

Testing Designs with Functional Simulation

After you have entered your design, you can either simulate or implement your design. 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. For Alliance Software Series users, details on functional simulation can be found in the CAE-specific interface user guide provided with your Xilinx interface. For Foundation users, refer to the *Foundation Series User Guide.*

Netlist Translation Program Overview

Two netlist translation programs allow you to read netlists into the Xilinx software tools. EDIF2NGD allows you to read an EDIF 2 0 0 (Electronic Data Interchange Format) file. XNF2NGD allows you to read an XNF (Xilinx Netlist Format). In Figure 2-3, these programs are contained within the "Netlist Translation" block. The NGDBuild program automatically invokes these programs as needed to convert your EDIF or XNF file to the required format for the Xilinx software tools.

You can find detailed descriptions of the EDIF2NGD, XNF2NGD, and NGDBuild programs in later chapters in this book, the *Development System Reference Guide*.

Design Implementation

Design implementation begins with the mapping or fitting of a logical design file to a specific device and is complete when the physical design has been successfully routed and a bitstream has been generated.

The following two figures give an overall view of the design implementation process for FPGAs and CPLDs.

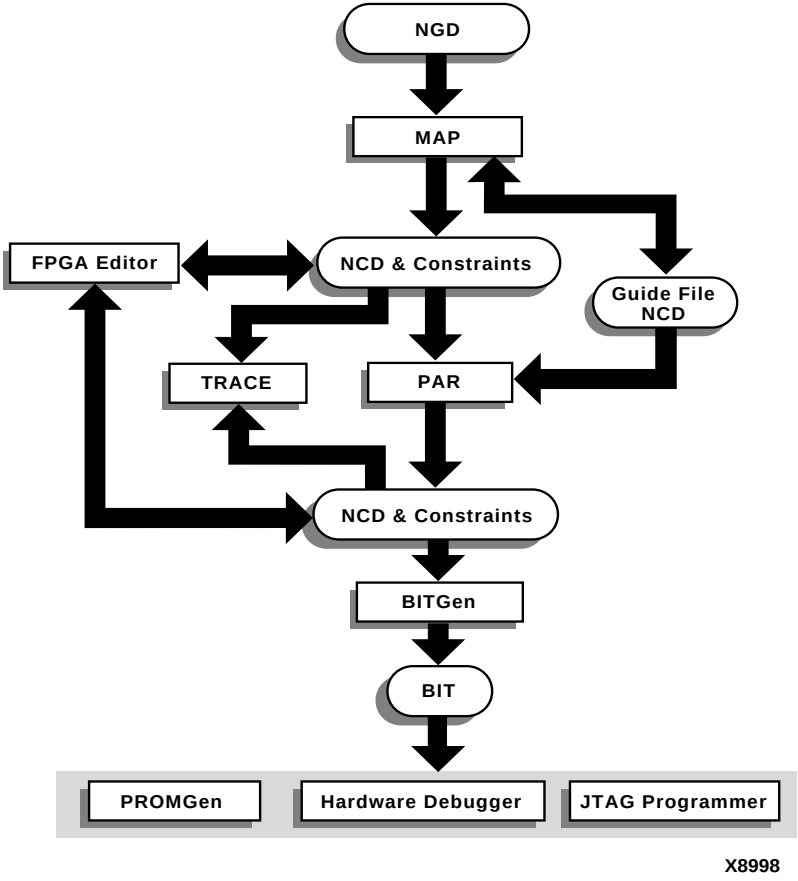


Figure 2-4 Design Implementation Flow (FPGAs)

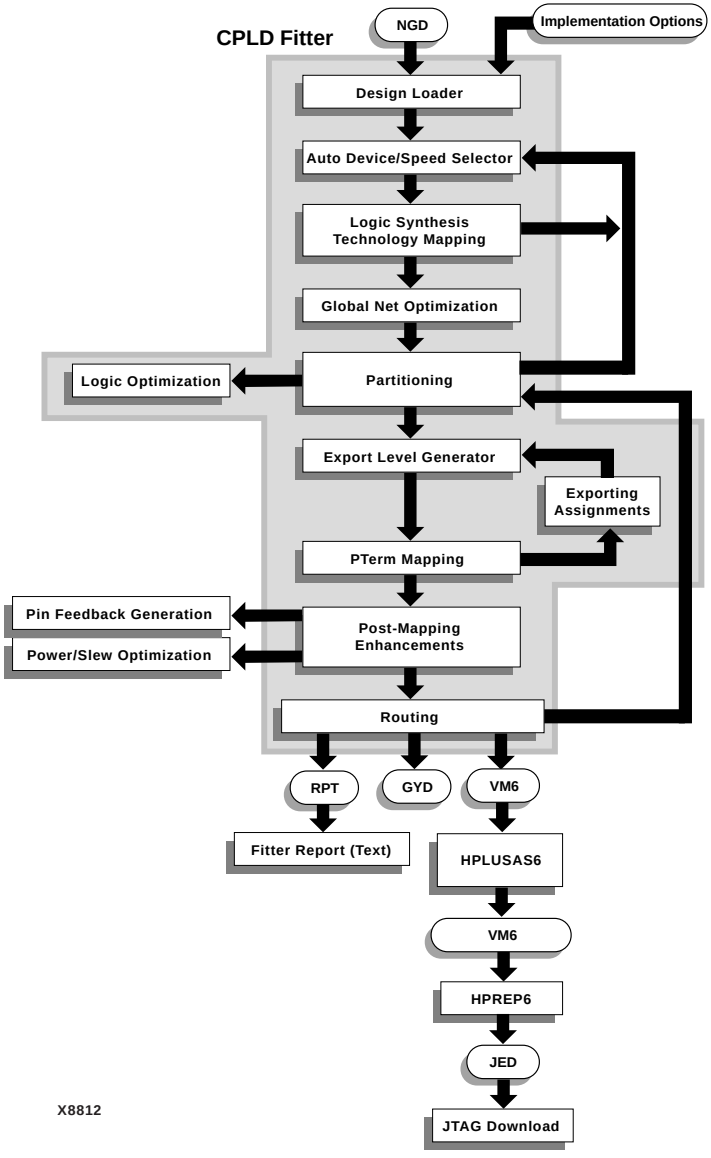


Figure 2-5 Design Implementation Flow (CPLDs)

FPGA Editor

The FPGA Editor is a graphical application used to display and configure your FPGA designs. You can perform the following functions with the FPGA Editor.

- Place and route critical components before running automatic place and route tools on an entire design
- Modify placement and routing manually; the editor allows both automatic and manual component placement and routing
- Read from and write to the Physical Constraints File (PCF) to create and modify constraints
- Verify timing against constraints
- Create physical macros (NMC files)

Editing operations performed within the FPGA Editor change the configuration of the design and also change the design database. Editing functions include selecting, adding, and deleting objects, viewing and changing object attributes, copying components, swapping components and net pins, placing components, and routing.

The FPGA Editor has its own set of commands, many of which can be customized to suit your work style. You can access these commands from the pull-down menus, user toolbar, command macros, hot keys and aliases, or by entering the commands on a command line within the FPGA Editor window.

In addition to customizing commands, other FPGA Editor tools may also be tailored to your needs. For example, you can do the following.

- Decide what commands are performed when the FPGA Editor window first opens
- Choose the colors for different objects (for example, components) and areas (for example, the push button panel or command area) in the FPGA Editor window
- Use command macros and hot keys

For more information, see the *FPGA Editor Guide*.

FPGA Design Techniques

The Xilinx FPGA architecture is best suited for synchronous design. Strict synchronous design ensures that all registers are driven from the same time base with no clock skew. The following sections outline several tips for producing high-performance synchronous designs.

Design Size and Performance

Information about design size and performance can help you to optimize your design. When you place and route your design, the resulting report files list the number of CLBs, IOBs, and other device resources available. A first pass estimate can be obtained by processing the design through the MAP program.

If you want to determine the design size and performance without running automatic implementation software, you can quickly obtain an estimate from a rough calculation based on the Xilinx FPGA architecture. See *The Programmable Logic Data Book* for more information on all Xilinx FPGA architectures.

Global Clock Distribution

Xilinx clock networks guarantee extremely small clock skew values. The following table lists the resources available for the Xilinx FPGA families.

Table 2-2 Global Clock Resources

FPGA Family	Resource	Number	Destination Pins
XC3000A / L XC3100A / L	GCLK ACLK	1 1	Clock Clock
XC4000E / L	BUFGP BUFGS	4 4	Clock, control, or certain input Clock, control, or certain input
XC4000EX / XL / XV / XLA	BUFG BUFGLS BUFGE BUFFCLK	8 8 8 4	Clock, control, or certain input Clock, control, or certain input Clock, control, or certain input I / O clock
XC5200	BUFG	4	Clock, control, or certain input
Spartan	BUFGP BUFGS	4 4	Clock, control, or certain input Clock, control, or certain input

Table 2-2 Global Clock Resources

FPGA Family	Resource	Number	Destination Pins
SpartanXL	BUFGLS	8	Clock, control, or certain input
Virtex, Virtex-E, Spartan-II	BUFG	4	Clock
Virtex-II	BUFGMUX	16	Clock

Note In certain devices families, global clock buffers are connected to control pin and logic inputs. If a design requires extensive routing, there may be extra routing delay to these loads.

Other Synchronous Design Considerations

Other considerations for achieving a synchronous design include the following.

- Use clock enables instead of gated clocks to control the latching of data into registers. See the figures in the following section.
- If your design has more clocks than the number of global clock distribution networks, try to redesign to reduce the number of clocks. Otherwise, put the clocks that have the lowest fanout onto normally routed nets, and specify a low MAXSKEW rating. Remember that a clock net routed through a normal net has skew.

Data Feedback and Clock Enable

The following figure shows a gated clock. The gated clock's corresponding timing diagram indicates that this implementation can lead to clock glitches; this can cause the flip-flop to clock at the wrong time.

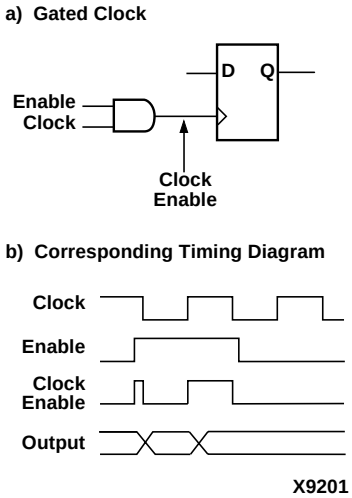


Figure 2-6 Gated Clock

The following figure shows a synchronous alternative to the gated clock using a data path. The flip-flop is clocked at every clock cycle and the data path is controlled by an enable. When the enable is Low, the multiplexer feeds the output of the register back on itself. When the enable is High, new data is fed to the flip-flop and the register changes its state. This circuit guarantees a minimum clock pulse width and it does not add skew to the clock. The XC4000, XC5200, Spartan-II, and Virtex flip-flops have a built-in clock-enable (CE).

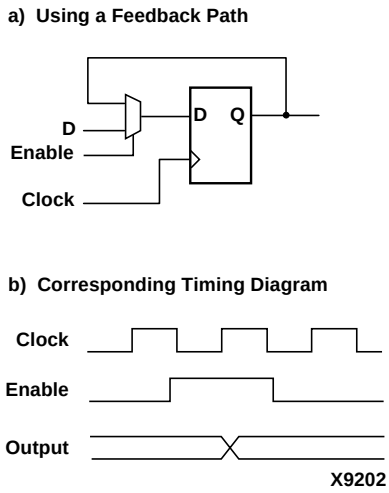


Figure 2-7 Synchronous Design Using Data Feedback

Counters

Cascading several small counters to create a larger counter is similar to a gated clock. For example, if two 8-bit counters are connected, the TC (terminal counter) of the first counter is a large AND function gating the second clock input. Using the CE input, you can create a synchronous design as shown in the following figure. In this case, the TC (terminal counter) of the first stage is connected directly to the CE of the second stage.

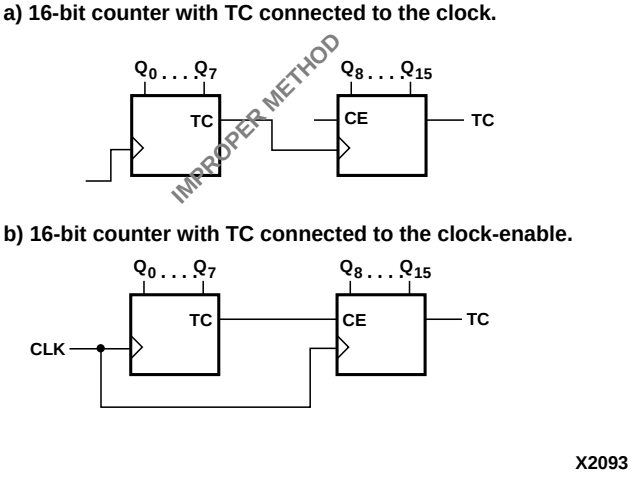


Figure 2-8 Two 8-Bit Counters Connected to Create a 16-Bit Counter

Design Verification

This section introduces design verification, which is the process of testing the functionality and performance of your design. The Xilinx Development System supports three complementary methods for design verification: simulation, static timing analysis, and in-circuit verification.

Overview

Design verification procedures should occur throughout your design process, as illustrated in the following figure.

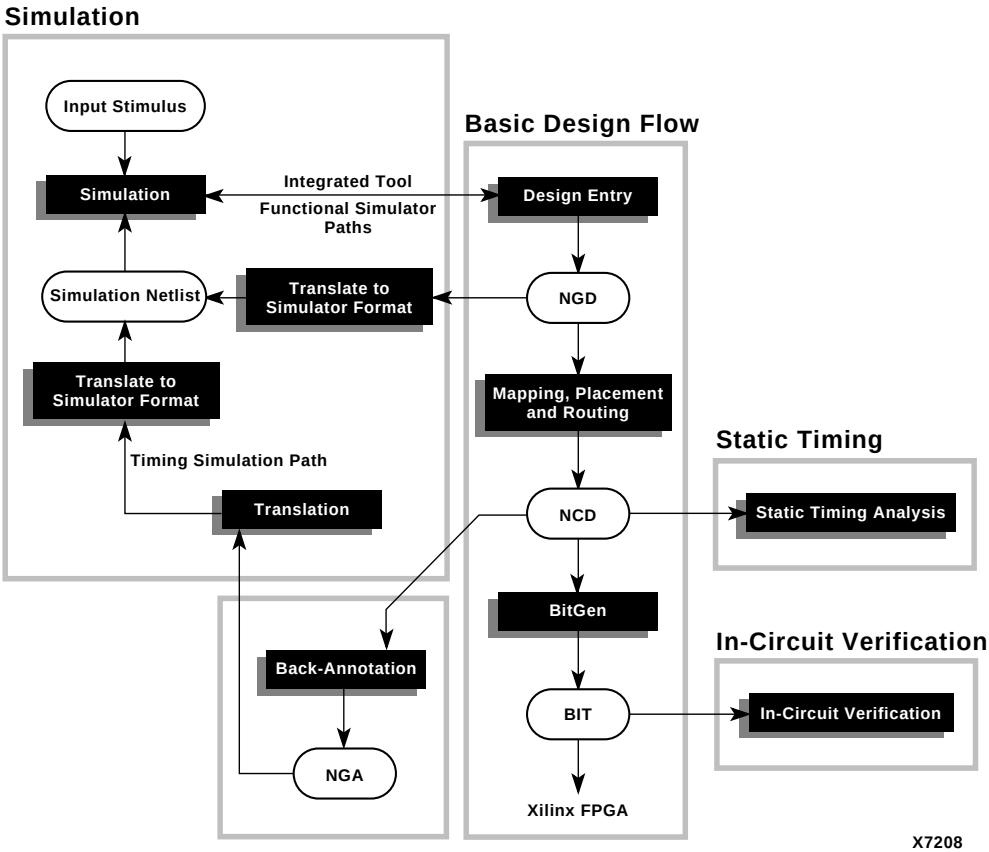


Figure 2-9 Three Verification Methods of the Design Flow (FPGAs)

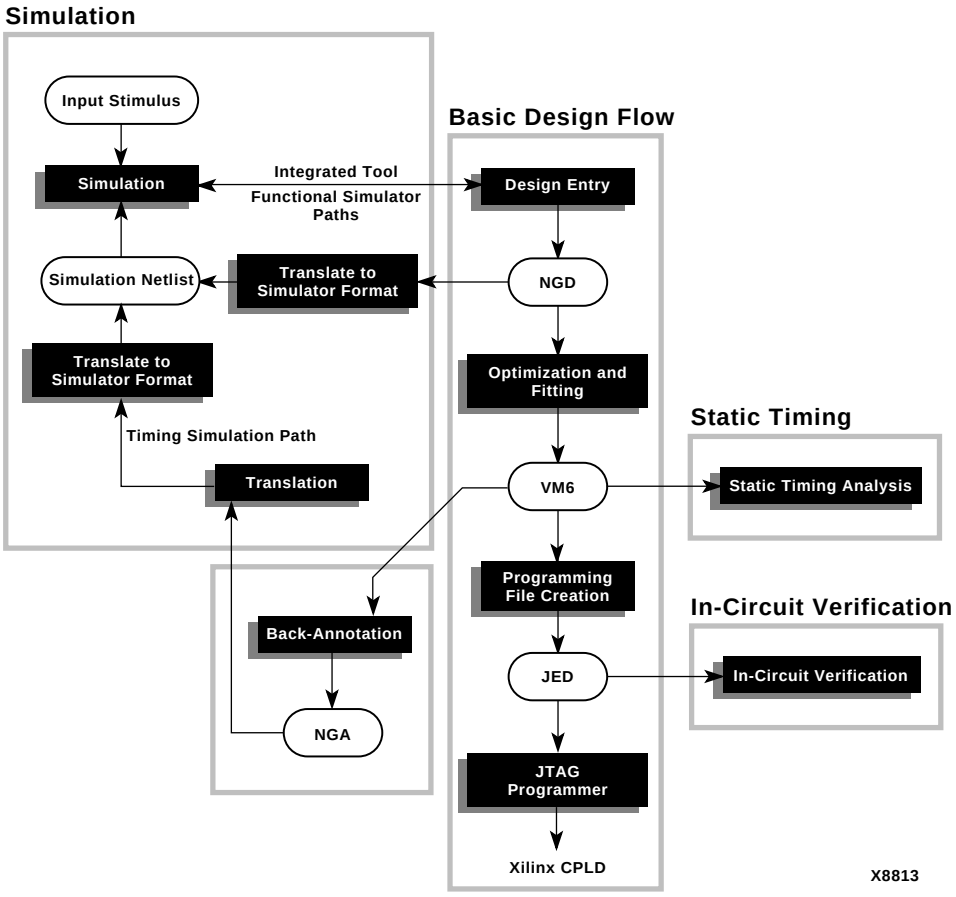


Figure 2-10 Three Verification Methods of the Design Flow (CPLDs)

You can verify Xilinx designs in three different ways.

- Simulation
- Static timing
- In-circuit verification

Each verification type uses different design tools, as shown in the following table.

Table 2-3 Verification Tools

Type of Verification	Tools
Simulation	Third Party Simulators (Integrated and Non-Integrated)
Static Timing	TRACE (Command Line) Timing Analyzer (GUI) Mentor Graphics® TAU and Viewlogic® BLAST software for use with the STAMP file format (for I/O timing verification only)
In-Circuit Verification	Design Rule Checker Download or XChecker Cable

Pre-Simulation Translation

Before simulation occurs, the physical design information must be translated and distributed back to the logical design. For FPGAs, this back-annotation process is done with a program called NGDAnno. For CPLDs, back-annotation is performed with the TSIM Timing Simulator. These programs create a database for the netlist writers, which translate the back-annotated information into a netlist format that can be used for simulation. The back-annotation flows are shown in the following figures.

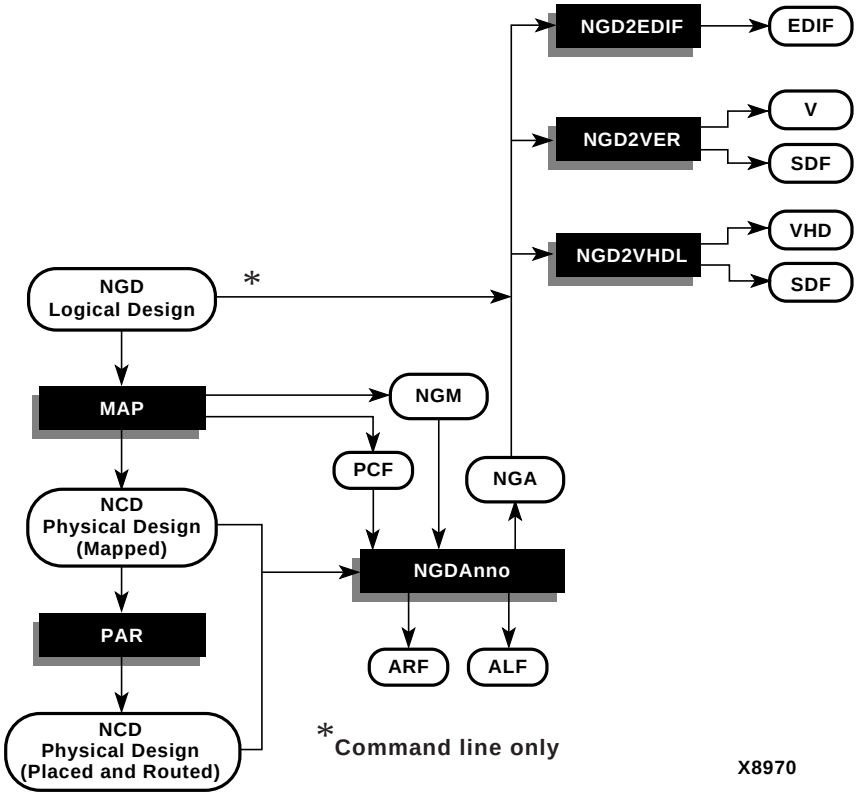


Figure 2-11 Back-Annotation (FPGAs)

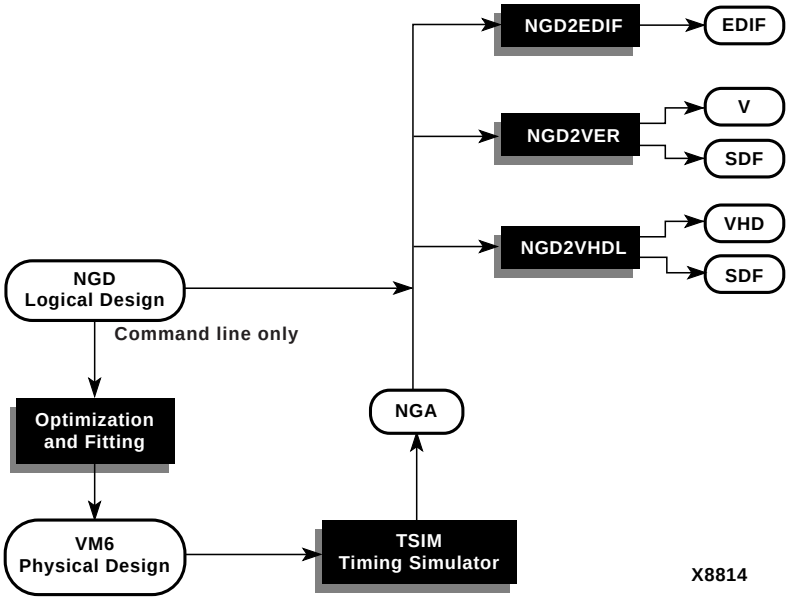


Figure 2-12 Back-Annotation (CPLDs)

Note The NGD2XNF program and the XNF output file format are not supported in this software release.

NGDAnno (FPGAs Only)

NGDAnno is a program that 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 with placement, routing, and timing information from the NCD file. This data is combined into a Native Generic Annotated (NGA) file. The NGA file is input to the appropriate netlist writer (NGD2EDIF, NGD2VER, or NGD2VHDL) which then converts the binary Xilinx database format back to an ASCII netlist.

Note Use caution when making changes to the functional behavior of your design. For example, if you make logical changes to an NCD design from within the FPGA Editor, NGDAnno will be unable to correlate the changed objects in the physical design with the objects in the logical design. It will then recreate the entire NGA design from the NCD and issue a warning indicating that the NCD is out of sync with the NGM.

An NCD file is input to the NGDAnno program. The NCD file can be a mapped-only design, or a partial or fully placed and routed design. An NGM file which is created by the mapper is an optional source of input.

The output of NGDAnno is an NGA file, which is a back-annotated NGD file. For details on NGDAnno refer to the "NGDAnno" chapter.

CPLD Command

The CPLD command automatically generates the NGA file unless the command is run with the –notsim option. For a description of the CPLD command and its options, refer to the "Fitter Command and Option Summary" appendix in the *CPLD Synthesis Design Guide*.

Netlist Writers

Netlist writers take the output of NGDAnno or the CPLD command 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. Following is a list of the supported netlist writers with descriptions of their input and output files.

- NGD2EDIF — takes an NGD or NGA file and translates it into an EDIF netlist.

  Output from the NGD2EDIF program is an EDN file, a netlist in EDIF format. The default EDN file produced by NGD2EDIF is generic. If you want to produce EDIF targeted to Mentor Graphics or Viewlogic, you must include the –v (vendor) option.

- NGD2VER — translates an NGD or NGA file into a Verilog netlist (V) file. If the input is an NGA file, NGD2VER also generates an SDF (Standard Delay Format) file.

The resulting V and SDF files have the same root name as the NGD or NGA file unless you specify otherwise.

The SDF file contains timing information intended solely for use with the Verilog file which was generated from the same NGA file. Do not attempt to use the SDF file in conjunction with the original Verilog netlist design or the product of another netlist writer.

- NGD2VHDL — translates an NGD or NGA file into a VHDL netlist (VHD). If the input file is an NGA file, NGD2VHDL also generate an SDF (Standard Delay Format) file. The VHD and SDF files have a time_sim root name by default.

  Note You can change the root name of your files using the Simulation Options dialog box in the Design Manager. See the Design Manager or Flow Engine online help for more information.

For more information on the Netlist Writers or NGDAnno, refer to later chapters in this manual.

Invoking Netlist Writer Programs

You can invoke any of the supported netlist writer programs (for example, NGD2VER, NGD2VHDL, or NGD2EDIF) from the UNIX or DOS command line. These programs and their options are described in later chapters of this book.

You can also set most of the netlist writer options using the Alliance Design Manager or Foundation Project Manager. To access these options, do the following.

Note The Foundation Series ISE Project Navigator runs the default settings for the netlist writer options. The netlist writer used is determined by the synthesis tool selected in the Project Properties dialog box. Right-click the Targeted Device and Synthesis Tool entry in the Process window and select **`Properties`** to open the Project Properties dialog box.

1. Do one of the following.
   - In the Design Manager, select **`Design`** → **`Options`**.
   - In the Project Manager, select **`Implementation`** → **`Options`**.

2. Select a simulation vendor from the Simulation drop-down list box.
3. Click **`Edit Options`**.

Note See the online help available from these tools for more information on each option.

Additional Translation Options

In addition to back-annotating a fully routed design, you can back-annotate a translated but unmapped design or a mapped but unrouted design for FPGAs. You can also create an output netlist to allow simulation of the design at the different stages of development in the Xilinx environment.

Pre-implementation Circuit Verification

For example, if you want to verify that the circuit logic is correct *before* you implement the design, you can use the data in a non-implemented NGD design as input to the netlist writers NGD2EDIF, NGD2VER, or NGD2VHDL. You can then run a simulation program on the resulting netlist.

Simulating Designs with Block Delays (FPGAs Only)

To simulate a design that contains only IOB and CLB block delays, you can take the NCD file produced by MAP and then run NGDAnno. Afterwards, run the appropriate netlist writer to generate a simulatable netlist.

Block delays are generally 50% of your path delay. Simulating with block delays is an imprecise method of determining whether your timing will be met before you actually place and route. (However, this simulation type is less time consuming than performing a full timing simulation.) The simulation process is shown in Figure 2-11.

Schematic-Based 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 schematic.

Simulation is performed using third-party tools that are linked to the Xilinx Development System. Use the various CAE-specific interface user guides, which cover the commands and features of the Xilinx-supported simulators, as your primary reference.

The software models provided for your simulation tools are designed to perform detailed characterization of your design. You can perform functional or timing simulation, as described in the following sections.

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.

Note It is usually faster and easier to correct design errors if you perform functional simulation early in the design flow.

The verification methods figures show the design flows for integrated and non-integrated simulation tools. Integrated tools such as Mentor or Viewlogic contain a built-in interface which links the simulator and a schematic editor, allowing the tools to use the same netlist. You can move directly from entry to simulation when using a set of integrated tools.

Functional simulation in schematic-based tools is usually performed immediately after design entry in the capture environment. The schematic capture tool requires a Xilinx Unified Library and the simulator requires a library if the tools are not integrated. Most of the schematic-based tools will require translation from their native database to XNF or EDIF for implementation. The return path from implementation is usually XNF or EDIF with certain exceptions where a schematic tool is tied to an HDL simulator.

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, and placed and routed for FPGAs or fitted for CPLDs. 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 can also determine whether or not the design contains set-up or hold violations.

To input timing information into your design, you must convert the routed NCD file into an NGA file. The resulting NGA file can then be translated by NGD2EDIF, NGD2VER, or NGD2VHDL. These netlist writers create suitable formats for various simulators.

Note Naming the nets during your design entry is important for both functional and timing simulation. This allows you to find the nets in the simulations more easily than looking for a machine-generated name.

HDL-Based Simulation

Xilinx supports functional and timing simulation of HDL designs at the following three points in the HDL design flow as shown in the following figure.

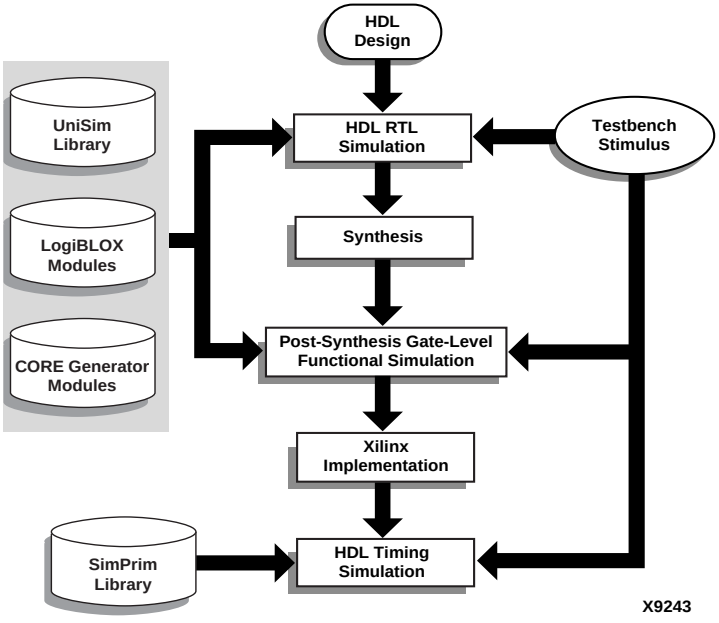


Figure 2-13 Simulation Points for HDL Designs

- Register Transfer Level (RTL) simulation which may include the following
  - Instantiated UniSim library components
  - LogiBLOX modules
  - LogiCORE models
- Post-synthesis functional simulation with one of the following.
  - Gate-level UniSim library components
  - Gate-level pre-route SimPrim library components
- Post-implementation back-annotated timing simulation with the following.
  - SimPrim library components
  - Standard Delay Format (SDF) file

The three primary simulation points can be expanded to allow for two additional post-synthesis simulations, as shown in the following table. These two additional points can be used when the synthesis tool either cannot write VHDL or Verilog, or if the netlist is not in terms of UniSim components.

Table 2-4 Five Simulation Points in HDL Design Flow

Simulation	UniSim	LogiBLOX Models	SimPrim	SDF
RTL	X	X		
Post-Synthesis	X	X		
Functional Post-NGDBuild (Optional)			X	
Functional Post-MAP (Optional)			X	X
Post-Route Timing			X	X

These simulation points are described in the “Simulation Points” section of the *Synthesis and Simulation Design Guide*.

The libraries required to support the simulation flows are described in detail in the “VHDL / Verilog Libraries and Models” section of the *Synthesis and Simulation Design Guide*. The new flows and libraries now support closer functional equivalence of initialization behavior

between functional and timing simulations. This is due to the addition of new methodologies and library cells to simulate GSR / GTS behavior.

It is important to address the built-in reset circuitry behavior in your designs starting with the first simulation to ensure that the simulations agree at the three primary points.

If you do not simulate GSR (Global Set / Reset) behavior prior to synthesis and place and route, your RTL and possibly post-synthesis simulations will not initialize to the same state as your post-route timing simulation. As a result, your various design descriptions are not functionally equivalent and your simulation results will not match.

In addition to the behavioral representation for GSR, you need to add a Xilinx implementation directive. This directive is used to specify to the place and route tools to use the special purpose GSR net that is pre-routed on the chip, and not to use the local asynchronous set / reset pins. Some synthesis tools can identify, from the behavioral description, the GSR net, and will place the STARTUP module on the net to direct the implementation tools to use the global network. However, other synthesis tools interpret behavioral descriptions literally, and will introduce additional logic into your design to implement a function. Without specific instructions to use device global networks, the Xilinx implementation tools will use general purpose logic and interconnect resources to redundantly build functions already provided by the silicon.

Even if GSR behavior is not described, the actual chip initializes during configuration, and the post-route netlist will have this net that must be driven during simulation. The “Understanding the Global Signals for Simulation” section of the *Synthesis and Simulation Design Guide* includes the methodology to describe this behavior, as well as the GTS (Global Tristate) behavior for output buffers.

For a complete discussion of GSR and GTS, refer to the “Defining Global Signals in VHDL” and “Defining Global Signals in Verilog” sections of the *Synthesis and Simulation Design Guide*.

Xilinx VHDL simulation supports the VITAL standard. This standard allows you to simulate with any VITAL-compliant simulator.

Built-in Verilog support allows you to simulate with the Cadence Verilog-XL and other compatible simulators. Xilinx HDL simulation

supports all current Xilinx FPGA and CPLD devices. Refer to the "VHDL / Verilog Libraries and Models" section for the list of supported VHDL and Verilog standards.

For information about CPLD HDL simulation refer to the "Simulating Your Design" chapter of the *CPLD Synthesis Design Guide*.

Static Timing Analysis With TRACE (FPGAs Only)

Static timing analysis is best for quick timing checks of a design after placement and routing is complete.

TRACE (Timing Reporter and Circuit Evaluator) is a Xilinx application program designed to provide static timing analysis and can be used to evaluate how well the place and route tools have met any input timing constraints.

By using TRACE, you can quickly check for timing problems in your FPGA design. You can also use TRACE to determine path delays in your design.

TRACE performs two major functions.

- Timing verification — the process of verifying that the design meets your timing constraints.
- Reporting — the process of enumerating input constraint violations and placing them into an accessible file. TRACE can be run on partially or completely placed and routed designs. The timing information reported issued by TRACE depends on the completeness of the placement and routing of the input design.

Within the Design Manager, TRACE is run using the Timing Analyzer. See the *Timing Analyzer Guide* for details.

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 with the Xilinx XChecker Cable, Parallel Cable III, or MultiLINX cable.

Note For information about Xilinx cables and hardware, see the *Hardware User Guide.*

Design Rule Checker (FPGAs Only)

Before generating the final bitstream, it is important to use the DRC option in BitGen to evaluate the NCD file for problems that could prevent the design from functioning properly. DRC is invoked automatically unless you use the –d option.

Xilinx Design Download Cables

Xilinx provides the Parallel Cable III, XChecker cable, or MultiLINX cable, depending on which development system you are using. To download your design, you must create a configuration bitstream.

For FPGAs, you can use the XChecker or MultiLINX cable to read back and verify configuration data. Detailed cable connection and daisy-chain information is provided in the *Hardware Debugger Guide.*

Note The Xilinx Parallel Cable III can be used for FPGA and CPLD design download and readback, but it does not have a design verification function.

With the XChecker cable, you can use the Hardware Debugger to take snapshots of the circuit at specific clock cycles. You can obtain these snapshots by performing serial readback of the nodes during in-circuit operation. With the Hardware Debugger software, you can speed up your analysis by limiting the readback bitstream to only those nodes and clock cycles in which you have interest.

You can also use the XChecker cable to probe your design after you download it. Probing internal nodes allows you to pinpoint the location of any design problems.

Use the XChecker cable when you do not want to specify additional IOBs and routing resources on your Xilinx FPGA for probing. This allows you to decide how you want to probe after you have downloaded your design.

The MultiLINX Cable is compatible in supporting readback and verify for all the FPGAs supported by the XChecker cable. In addition to the supporting legacy devices, the MultiLINX Cable supports the devices that were not supported by the XChecker cable. Supported devices include those devices in the 4000E, 4000XL, and Spartan families whose BIT file size is more than 256K bits. The MultiLINX Cable also supports readback and verify for Virtex device families.

Note Debug is not available with the MultiLINX Cable in this release.

Chapter 3

PARTGEN

This program is compatible with the following Xilinx devices.

- Spartan™ / XL
- Virtex™
- XC9500™ / XL
- XC4000E™ / L / EX / XL / XV / XLA
- XC3000A™ / L
- XC3100A™ / L
- XC5200™

This chapter describes PARTGEN. The chapter contains the following sections.

- "PARTGEN"
- "PARTGEN Syntax"
- "PARTGEN Files"
- "PARTGEN Options"
- "Partlist.xct File Contents"

PARTGEN

The PARTGEN command displays various levels of information about installed Xilinx devices and families depending on which options are selected.

PARTGEN Syntax

Following is the syntax for PARTGEN.

partgen [*options*]

Options can be any number of the options listed in the "PARTGEN Options" section. They do not need to be listed in any particular order. Separate multiple options with spaces.

PARTGEN Files

The following subsections describe the input and output files for PARTGEN.

Input Files

PARTGEN does not have any user input files.

Output Files

PARTGEN creates the output file, partlist.xct. This output file is only generated when using the –p and –v options. See the "Partlist.xct File Contents" section for a detailed description.

The –p and –v options also generate package files. These files correlate IOBs with output pin names. The –p option generates a three column entry describing the pins. The –v option adds four more columns of descriptive pin information.

For example, the command partgen –p xc4003 generates the package files: 4003cb100.pkg, 4003pc84.pkg, 4003pq100.pkg, 4003cq100.pkg, and 4003pg120.pkg. Following is a portion of the package file for the xc4003cb100.

```
package 4003cb100
pin PAD1 P99
pin PAD2 P98
pin PAD3 P97
pin PAD4 P96
pin PAD5 P95
pin PAD6 P94
pin PAD7 P93
pin PAD8 P92
pin PAD9 P91
```

.
.
.

The first column contains either “pin” (user accessible pin) or “pkpin” (dedicated pin). The second column specifies the pin name. For user accessible pins, the name of the pin is the bonded pad name associated with an IOB on the device, or the name of a multipurpose pin. For dedicated pins, the name is either the functional name of the pin, or N.C. specifying No Connection. The third column specifies the package pin.

The command partgen –v generates package (.pkg) files and generates a seven column entry describing the pins. The first three columns are described above.

The fourth column, IO_BANK, is a positive integer associated with a bank, or -1 for no bank association. The fifth column, specifying function name, consists of a string indicating how the pin is used. If the pin is dedicated, then the string will indicate a specific function. If the pin is a generic user pin, the string will be “IO.” If the pin is multipurpose, an underscore separated set of characters will make up the string. The sixth column indicates the closest CLB row or column to the pin, and appears in the form R[0-9]C[0-9]. The seventh column is composed of a string for each pin associated with a LVDS IOB. The string consists of and index and the letter M or S. Index values will go from 0 to the number of LVDS pairs. The value for a non-LVDS pin will default to N.A. The following are examples of the verbose pin descriptors in partgen.

```
pkpin N.C. D9 -1 N.C.N.A. N.A.
pkpin DONE M12 -1 DONE N.A.N.A.
pkpin VCCO N1 -1 VCCO N.A.N.A.
pin PAD2 B3 0 IO R0C0 0S
pin PAD17 AB23 7 IO_TDO R6C48 N.A.
```

PARTGEN Options

Following is a description of the command line options and how they affect the behavior of PARTGEN. With no options, PARTGEN prints out the following usage message.

```
partgen - Print out the list of supported parts for the installed
architectures.
-i: print a list of devices, packages, and speeds that are
installed.
-arch: print a list of devices, packages, and speeds for the
specified
    architecture (if it is installed).
-p: generate a partlist.xct. If no architecture, device or part
    is specified, print all information to parlist.xct. Otherwise
generate
    data only for the family members specified.
-v: generate a verbose partlist.xct. If no architecture, device, or
device
    package is specified, print all information to parlist.xct,
Otherwise
    generate data only for the family members specified.
NOTE: -v and -p options are mutually exclusive.
```

–arch (Print Information for Specified architecture)

-arch *architecture_name*

The –arch option prints a list of devices, packages, and speeds for a specified architecture that has been installed.

Valid entries for *architecture_name* are as follows:

- Spartan
- SpartanXL
- Virtex
- XC9500
- XC9500XL
- XC4000
- XC4000EX
- XC4000XL
- XC4000XV
- XC4000XLA
- XC3000

- XC3100
- XC5200

For example, entering the command partgen -arch spartan2 displays the following information.

```
2s100                SPEEDS:-6 -5 -4
      x2s100
2s15                 SPEEDS:-6 -5 -4
      x2s15
2s150                SPEEDS:-6 -5 -4
      x2s150
2s30                 SPEEDS:-6 -5 -4
      x2s30
2s50                 SPEEDS:-6 -5 -4
      x2s50
```

–i (Print a List of Devices, Packages, and Speeds)

The –i option prints out a list of devices, packages, and speeds that have been installed. Following is a portion of a sample display.

```
s05                SPEEDS:     -4     -3
      PC84
      VQ100
s10                SPEEDS:     -4     -3
      PC84
      VQ100
      TQ144
s20                SPEEDS:     -4     -3
      VQ100
      TQ144
      PQ208
s30                SPEEDS:     -4     -3
      VQ100
      TQ144
      PQ208
      PQ240
      BG256
s40                SPEEDS:     -4     -3
      PQ208
      PQ240
```

```
      BG256
2s100             SPEEDS:    -6    -5    -4
      x2s100
2s15              SPEEDS:    -6    -5    -4
      x2s15
.
.
.
```

–p (Creates Package file and Partlist.xct File)

-p *name*

The –p option generates a partlist.xct file for the specified name and also creates package files. All files are placed in the current working directory. Valid name entries include architectures, devices, and parts. Following are example command line entries of each type.

-p xc4000 (Architecture)

-p xc4003 (Device)

-p 4003CB100 Part)

If no architecture, device, or PART is specified with the –p option, detailed information for every installed device is submitted to the partlist.xct file.

The –p option generates more detailed information than the -arch options but less information that the –v option. The –p and –v options are mutually exclusive, that is, you can specify one or the other but not both.

Following is an example display for the command **partgen -p 4036EXHQ240**.

```
part4000 4036EXhq240 NA.die 4036hq240.pkg \
        TDI=PAD284 TCK=PAD283 TMS=PAD268 \
        NCLBROWS=36 NCLBCOLS=36 STYLE=XC4000EX \
        EDGE_DECODERS=4 \
        IN_FF_PER_IOB=1 OUT_FF_PER_IOB=1 \
        NFRAMES=1775 NBITSPERFRAME=462
```

For a description of the entries in the partlist.xct file, see the "Partlist.xct File Contents" section.

–v (Creates Packages and Partlist.xct File)

–v name

The –v option generates a partlist.xct file for the specified name and also creates packages files. Valid name entries include architectures, devices, parts. Following are example command line entries of each type.

partgen –v xc4000 (Architecture)

partgen –v xc4003 (Device)

partgen –v 4003CB100 (Part)

If no architecture, device, or part is specified with the –v option, information for every installed device is submitted to the partlist.xct file.

The –v option generates more detailed information than the –p option. The –p and –v options are mutually exclusive, that is, you can specify one or the other but not both.

Following is an example display for the command **partgen -v 4036EXHQ240**.

```
part4000 4036EXhq240 NA.die 4036hq240.pkg \
      TDI=PAD284 TCK=PAD283 TMS=PAD268 \
      NCLBROWS=36 NCLBCOLS=36 STYLE=XC4000EX \
      EDGE_DECODERS=4 \
      IN_FF_PER_IOB=1 OUT_FF_PER_IOB=1 \
      NPADS_PER_ROW=2 NPADS_PER_COL=2 \
      NFRAMES=1775 NBITSPERFRAME=462 \

      NIOBS=288 NBIOBS=193 \
      SLICES_PER_CLB=1 \
      FFS_PER_SLICE=2 \
      LATCHES_PER_SLICE=TRUE \
      LUT_NAME=F LUT_SIZE=4 LUT_NAME=G LUT_SIZE=4 \
      LUT_NAME=H LUT_SIZE=3 \NUM_GLOBAL_BUFFERS=8 \
      BUFGLS_NNW=PAD1 BUFGLS_NNE=PAD71 BUFGLS_SSW=PAD216
BUFGLS_SSE=PAD145        BUFGLS_WNW=PAD288        BUFGLS_ENE=PAD73
BUFGLS_WSW=PAD217
BUFGLS_ESE=PAD143 \
      NUM_TBUFS_PER_ROW=76\
      NUM_CARRY_ELEMENTS_PER_SLICE=2\
```

```
      SPEEDGRADE=-2         LUTDELAY=1500         IOB_IN_DELAY=1710
IOB_OUT_DELAY=7310 \
      SPEEDGRADE=-3         LUTDELAY=1700         IOB_IN_DELAY=1810
IOB_OUT_DELAY=7820 \
      SPEEDGRADE=-4                                   LUTDELAY=2040
IOB_IN_DELAY=2170IOB_OUT_DELAY=9380
```

For a description of the entries in the partlist.xct file, see the "Partlist.xct File Contents" section.

Partlist.xct File Contents

The partlist.xct file contains detailed information about architectures and devices.

The partlist.xct file is a series of part entries. There is one entry for every part supported in the installed software. The following subsections describe the information contained in the partlist.xct file.

Header

The first part is a header for the entry. The format of the entry looks like the following.

```
part architecture family partname diename package-filename
```

Following is an example for the XC4036EXhq240.

```
part4000 4036EXhq240 NA.die 4036hq240.pkg
```

Device Attributes

The header is followed by a list of device attributes. Not all attributes are applicable to all devices.

- BSCAN pin mappings: TDK=PAD# TDI=PAD# TMS=PAD#
- CLB row and column sizes: NCLBROWS=# NCLBCOLS=#
- Sub-family designation: STYLE=sub_family
  (For example, STYLE = XC4000EX)
- Width of the edge decoder (found in the XC5200 and XC4000 families): EDGE_DECODER=#
- Input registers: IN_FF_PER_IOB=#

- Output registers: OUT_FF_PER_IOB=#
- Number of pads per row and per column: NPADS_PER_ROW=# NPADS_PER_COL=#
- Bitstream information:
  - Number of frames: NFRAMES=#
  - Number bits / frame: NBITSPERFRAME=#

The preceding bulleted items display for both the -p and -v options. The following bulleted items only display when using the -v option.

- Number of IOBS in device: NIOBS=#
- Number of bonded IOBS: NBIOBS=#
- Slices per CLB: SLICES_PER_CLB=#

  For slice-based architectures; for example. virtex.

  (For non-slice based architectures, assume one slice per CLB)
- Flip-flops for each slice: FFS_PER_SLICE=#
- Latches for each slice: LATCHES_PER_SLICE={**TRUE** | **FALSE**}
- LUTs in a slice: LUT_NAME=name LUT_SIZE=#
- Number of global buffers: NUM_GLOBAL_BUFFERS=#

  (The number of places where a buffer can drive a global clock combination)
- External Clock IOB pins:
  - For the XC3000 family: TCLKIOB=PAD# BCLKIOB=PAD#
  - For the XC4000 / XC4000E family:

    ```
    BUFGP_TL=PAD#, BUFGP_BL=PAD#,
    BUFGP_BR=PAD#, BUFGP_TR=PAD#,
    BUFGS_TL=PAD#, BUFGS_BL=PAD#,
    BUFGS_BR=PAD#, BUFGS_TR=PAD#
    ```

  - For the XC4000EX / XC4000XL / XC4000XLA / XC4000XV / SpartanXL families:

    ```
    BUFGLS_NNW=PAD#,
    BUFGLS_WNW=PAD#,
    BUFGLS_NNE=PAD#,
    ```

```
BUFGLS_ENE=PAD#,
BUFGLS_SSW=PAD#,
BUFGLS_WSW=PAD#,
BUFGLS_SSE=PAD#,
BUFGLS_ESE=PAD#
```

- For the XC5200 family:

  ```
  BUFG_TL=PAD#, BUFG_TR=PAD#,
  BUFG_BL=PAD#, BUFG_BR=PAD#
  ```

- For the Virtex families and Spartan2:

  ```
  GCLKBUF0=PAD#, GCLKBUF1=PAD#,
  GCLKBUF2=PAD#, GCLKBUF3=PAD#
  ```

- Oscillator pins for the XC3000 family:

  ```
  OSCIOB1=PAD#, OSCIOB2=PAD#
  ```

- Block RAM:

  ```
  NUM_BLK_RAMS=#
  BLK_RAM_COLS=# BLK_RAM_COL0=# BLK_RAMCOL1=#
  BLK_RAM_COL2=# BLK_RAM_COL_3=#
  BLK_RAM_SIZE=4096x1 BLK_RAM_SIZE=2048x2
  BLK_RAM_SIZE=512x8 BLK_RAM_SIZE=256x16
  ```

  Block RAM locations are given with reference to CLB columns. In the following example, Block RAM 5 is positioned in CLB column 32.

  ```
  NUM_BLK_RAMS=10 BLK_RAM_COL_5=32
  BLK_RAM_SIZE=4096X1
  ```

- Select RAM:

  ```
  NUM_SEL_RAMS=#   SEL_RAM_SIZE=#X#
  ```

- Select Dual Port RAM:

  ```
  SEL_DP_RAM={TRUE|FALSE}
  ```

  This field indicates whether the select RAM can be used as a dual port ram. The assumption is that the number of addressable elements is reduced by half, that is, the size of the select RAM in Dual Port Mode is half that indicated by SEL_RAM_SIZE.

- Speed grade information: SPEEDGRADE=#

- Typical delay across a LUT for each speed grade: LUTDELAY=#
- Typical IOB input delay: IOB_IN_DELAY=#
- Typical IOB output delay: IOB_OUT_DELAY=#
- Maximum LUT constructed in a slice:

  `MAX_LUT_PER_SLICE=#`

  (From all the LUTs in the slice)
- Max LUT constructed in a CLB: MAX_LUT_PER_CLB=#

  (This field describes how wide a LUT can be constructed in the CLB from the available LUTs in the slice.)
- Number of internal tristate buffers in a device: NUM_TBUFFS=#

Chapter 4

NGDBuild

This program is compatible with the following families.

- Spartan / XL / -II
- Virtex / -E / -II
- XC9500 / XL / XV
- XC4000E / L / EX / XL / XV / XLA
- XC3000A / L
- XC3100A / L
- XC5200

This chapter describes the NGDBuild program. The chapter contains the following sections.

- "NGDBuild"
- "NGDBuild Syntax"
- "NGDBuild Files"
- "NGDBuild Options"
- "Netlister Launcher"
- "File Names and Locations"

NGDBuild

NGDBuild performs all the steps necessary to read a netlist file in XNF or EDIF 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 NGD (Native Generic Database) primitives and a description in terms of the original hierarchy expressed in the input netlist. The output NGD file can be mapped to the desired device family.

The following figure is a simplified drawing of the design flow through NGDBuild. NGDBuild invokes other programs and generates intermediate (NGO) files that are not shown in the drawing. For a complete description of how NGDBuild works, see the "EDIF2NGD, XNF2NGD, and NGDBuild" appendix.

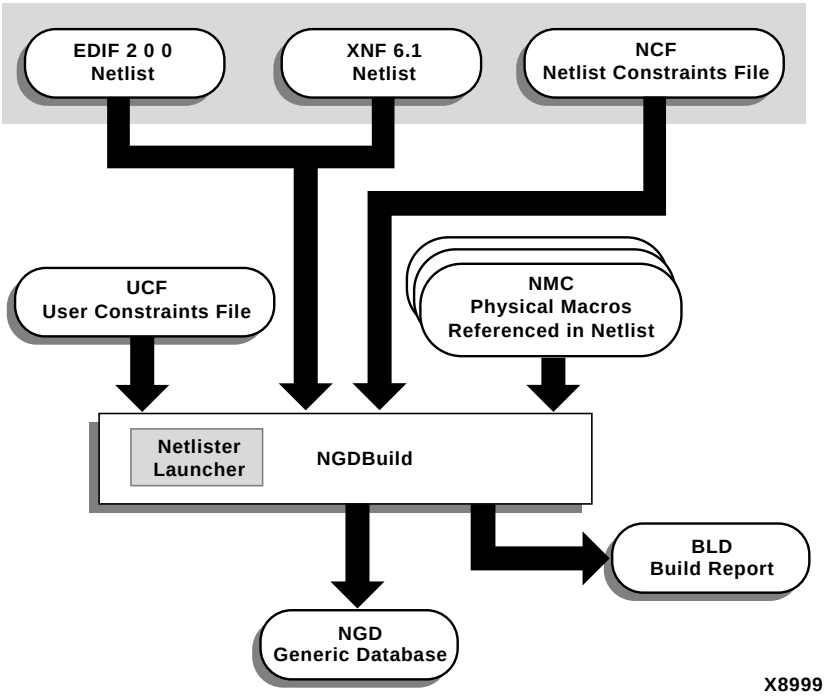


Figure 4-1 NGDBuild Design Flow

Converting a Netlist to an NGD File

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

1. Reads the source netlist

   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 (NMC files), and behavioral models.

3. Checks the design by running a Logical DRC (Design Rule Check) on the converted design

   The Logical DRC is a series of tests on the logical design. It is described in the "Logical Design Rule Check" chapter.

4. Writes an NGD file as output

Note This procedure, the Netlister Launcher, and the netlist reader programs are described in more detail in the "EDIF2NGD, XNF2NGD, and NGDBuild" appendix.

NGDBuild Syntax

The following command reads the design into the Xilinx Development system and converts it to an NGD file.

ngdbuild [*options*] *design_name* [*ngd_file*[**.ngd**]]

options can be any number of the NGDBuild options listed in the "NGDBuild Options" section. They do not need to be listed in any particular order. Separate multiple options with spaces.

design_name is the top-level name of the design file you want to process. To ensure the design processes correctly, specify a file extension for the input file. Use one of the legal file extensions specified in the "Input Files" section. Using an incorrect or nonexistent file extension causes NGDBuild to fail without creating an NGD file. If you use

an incorrect file extension, NGDBuild may issue an “unexpanded” error.

ngd_file[**.ngd**] is the output file in NGD format. The output file name, its extension, and its location are determined as follows.

- If you do not specify an output file name, the output file has the same name as the input file, with an .ngd extension.
- If you specify an output file name with no extension, NGDBuild appends the .ngd extension to the file name.
- If you specify a file name with an extension other than .ngd, you get an error message and NGDBuild does not run.
- If the output file already exists, it is overwritten with the new file.

NGDBuild Files

This section describes the NGDBuild input and output files.

Input Files

The input files to NGDBuild are the following.

- Design file—The input design can be an XNF or EDIF 2 0 0 netlist. If the input netlist is in another format that the Netlister Launcher recognizes, the Netlister Launcher invokes the program necessary to convert the netlist to EDIF or XNF format, then invokes the appropriate netlist reader, EDIF2NGD or XNF2NGD.

  With the default Netlister Launcher options, NGDBuild recognizes and processes files with the extensions shown in the following table. NGDBuild searches the top-level design netlist directory for a netlist file with one of these extensions in the order shown in this table. For example, NGDBuild searches for an XTF file first. If one is not found, it then searches for an XG file and so forth.

Netlist Type	Recognized Extensions
XNF	.xtf, .xg, .xff, .sxnf, .xnf
EDIF	.sedif, .edn, .edf, .edif
PLD	.pld

Note Remove all out of date netlist files from your directory. Obsolete netlist files may cause errors in NGDBuild.

- UCF file—The User Constraints File is an ASCII file that you create. You can create this file using the Constraints Editor. See the *Constraints Editor Guide* for more information. The file contains timing and layout constraints that affect how the logical design is implemented in the target device. The constraints in the file are added to the information in the output NGD file.

  By default, NGDBuild reads the constraints in the UCF file automatically if the UCF file has the same base name as the input design file and a .ucf extension. You can override the default behavior and specify a different constraints file by entering a –uc option to the NGDBuild command line. See the "–uc (User Constraints File)" section for more information.

- NCF file—The Netlist Constraints File is produced by a CAE vendor toolset. This file contains constraints specified within the toolset. The netlist reader invoked by NGDBuild reads the constraints in this file if the NCF file has the same name as the input netlist file. It adds the constraints to the intermediate NGO file and the output NGD file.

  Note If the NGO file for a netlist file is up to date, NGDBuild looks for an NCF file with the same base name as the netlist in the netlist directory and compares the timestamp of the NCF file against that of the NGO file. If the NCF file is newer, XNF2NGD or EDIF2NGD is run again. However, if an NCF file existed on a previous run of NGDBuild and the NCF file was deleted, NGDBuild does not detect that XNF2NGD or EDIF2NGD must be run again. In this case, you must use the –nt on option to force a rebuild. See the "–nt (Netlist Translation Type)" section for more information.

- NGC file—This binary file contains the implementation of a module in the design. If an NGC file exists for a module, NGDBuild reads this file directly, without looking for a source EDIF or XNF netlist. In HDL design flows, LogiBLOX creates an NGC file to define each module.

- NMC files—These physical macros are binary files that contain the implementation of a physical macro instantiated in the design. NGDBuild reads the NMC file to create a behavioral simulation model for the macro.

- MEM files—These LogiBLOX memory definition files are text files that define the contents of LogiBLOX memory modules. NGDBuild reads MEM files in design flows where LogiBLOX does not create NGC files directly. See the "Module Descriptions" chapter of the *LogiBLOX Guide* for details.

Unless a full path is provided to NGDBuild, it searches for netlist, NGC, NMC, and MEM files in the following locations.

- The working directory from which NGDBuild was invoked
- The path specified for the top-level design netlist on the NGDBuild command line
- Any path specified with the –sd switch on the NGDBuild command line

Output Files

Output from NGDBuild consists of the following files.

- NGD file—This binary file contains a logical description of the design in terms of both its original components and hierarchy and the NGD primitives to which the design reduces.
- BLD file—This build report file contains information about the NGDBuild run and about the subprocesses run by NGDBuild. Subprocesses include EDIF2NGD, XNF2NGD, and programs specified in the URF file. The BLD file has the same root name as the output NGD file and a .bld extension. The file is written into the same directory as the output NGD file.

Note If you attach a pull-up or pull-down property on a pad net in your UCF file, a comment in the BLD file indicates that NGDBuild added a pull-up or pull-down instance to the net.

Intermediate Files

NGO files—(Not shown in Figure 4-1) These binary files contain a logical description of the design in terms of its original components and hierarchy. These files are created when NGDBuild reads the input netlist. If these files already exist, NGDBuild reads the existing files. If these files do not exist or are out of date, NGDBuild creates them.

NGDBuild Options

This section describes NGDBuild command line options.

–a (Add PADs to Top-Level Port Signals)

If the top-level input netlist is in EDIF format, the –a option causes NGDBuild to add a PAD symbol to every signal that is connected to a port on the root-level cell. This option has no effect on lower-level netlists or on a top-level XNF netlist.

Whether you need to use –a depends on the behavior of your third-party EDIF writer. If your EDIF writer treats pads as instances (like other library components), you should not use –a. If your EDIF writer treats pads as hierarchical ports, you should use –a to infer actual pad symbols. If you do not use –a where necessary, logic may be improperly removed during mapping.

For EDIF files produced by Mentor Graphics and Cadence, the –a option is set automatically; you do *not* have to enter –a explicitly for these vendors.

Note The NGDBuild –a option corresponds to the EDIF2NGD –a option. If you run EDIF2NGD on the top-level EDIF netlist separately, rather than allowing NGDBuild to run EDIF2NGD, you should use the two –a options consistently.

–dd (Destination Directory)

-dd *ngo_directory*

The –dd option specifies the directory for intermediate files (design NGO files and netlist files). If the –dd option is not specified, files are placed in the current directory.

–f (Execute Commands File)

-f *command_file*

The –f option executes the command line arguments in the specified *command_file*. For more information on the –f option, see the "–f Option" section of the "Introduction" chapter.

–i (Ignore UCF File)

The –i option ignores the UCF file in the top-level design netlist directory. If –i is not specified, NGDBuild checks for a UCF file and, if one is detected, loads it.

Note If you use this option, do not use the –uc option.

–l (Libraries to Search)

`-l` *`libname`*

The –l option indicates the list of libraries to search when determining what library components were used to build the design. This option is passed to the appropriate netlist reader. The information allows NGDBuild to determine the source of the design's components so it can resolve the components to NGD primitives.

You can specify multiple libraries by entering multiple **–l** *libname* entries on the NGDBuild command line.

The allowable entries for *libname* are the following.

- **`xilinxun`** `(Xilinx Unified library)`
- **`synopsys`**

Note You do not have to enter xilinxun with a –l option. The Xilinx Development System tools automatically access these libraries. In cases where NGDBuild automatically detects Synopsys designs (for example, the netlist extension is .sxnf or .sedif), you do not have to enter synopsys with a –l option.

–modular initial (Initial Budgeting of Modular Design)

The –modular initial option sets up the initial budgeting for the modular design session. NGDBuild generates an NGO and NGD file for the top-level design with all of the instantiated modules represented as unexpanded blocks. This NGD file cannot be mapped but can be used with the Floorplanner and Constraints Editor tools.

For more information on modular design, see `http://support.xilinx.com/xapp/xapp404.pdf`.

–modular module (Active Module Implementation)

–modular module -active *active_module_name*

The –modular module option saves the name of the active module in the generated NGD file for later processing in a modular design session.

–modular assemble (Module Assembly)

–modular assemble -pimpath *pim_directory_path*
-use_pim *pim_module_name*

The –modular assemble option links the Physically Implemented Modules (PIMs) to the top-level design. Use the –pimpath sub-option to specify the directory that contains the PIMs. Use the –usepim sub-option to instantiate one or many PIMs in the top-level design. Following is an example of how to use the –usepim sub-option to specify two PIMs.

–modular assemble -pimpath *pimpath* **-use_pim** *pim_module_name1* **-use_pim** *pim_module_name2*

–nt (Netlist Translation Type)

–nt {**timestamp** | **on** | **off**}

The –nt option determines how timestamps are treated by the Netlister Launcher when it is invoked by NGDBuild. A timestamp is information in a file that indicates the date and time the file was created. The timestamp option (which is the default if no –nt option is specified) has the Netlister Launcher perform the normal timestamp check and update NGO files according to their timestamps. The on option translates netlists regardless of timestamps (rebuilding all NGO files), and the off option does not rebuild an existing NGO file, regardless of its timestamp.

–p (Target Architecture)

–p *part*

The –p option specifies the part into which the design is implemented.The –p option can specify an architecture only, a complete part specification (device, package, and speed), or a partial specification (for example, device and package only).

The syntax for the –p option is described in the "Part Numbers in Commands" section of the "Introduction" chapter. Examples of part entries are **XCV50-TQ144** and **XCV50-TQ144-5**.

When you specify the *part*, the NGD file produced by NGDBuild is optimized for mapping into that architecture.

You do not have to specify a –p option if your NGO file already contains information about the desired vendor and family (for example, if you placed a PART property in a schematic or a CONFIG PART statement in a UCF file). However, you can override the information in the NGO file with the –p option when you run NGDBuild.

–r (Ignore LOC Constraints)

The –r option eliminates all location constraints (LOC=) found in the input netlist or UCF file. Use this option when you migrate to a different device or architecture, because locations in one architecture may not match locations in another.

Note If you previously ran NGDBuild on your design and NGO files are present, you must use the –nt on option the first time you use –r. This forces a rebuild of the NGO files, allowing NGDBuild to run EDIF2NGD or XNF2NGD to remove location constraints.

–sd (Search Specified Directory)

-sd *search_path*

The –sd option adds the specified *search_path* to the list of directories to search when resolving file references (that is, files specified in the schematic with a FILE=*filename* property) and when searching for netlist, NGO, NGC, NMC, and MEM files. You do not have to specify a search path for the top-level design netlist directory, because it is automatically searched by NGDBuild.

The *search_path* must be separated from the –sd by spaces or tabs (for example, **–sd designs** is correct, **–sddesigns** is not).

You can specify multiple –sd options on the command line. Each must be preceded with –sd; you cannot combine multiple *search_path* specifiers after one –sd. For example, the following syntax is not acceptable.

```
-sd /home/macros/counter /home/designs/pal2
```

The following syntax is acceptable.

```
-sd /home/macros/counter -sd /home/designs/pal2
```

–u (Allow Unexpanded Blocks)

In the default behavior of NGDBuild (without –u option), NGDBuild generates an error if a block in the design cannot be expanded to NGD primitives. If this error occurs, an NGD file is not written.

If you enter the –u option, NGDBuild generates a warning instead of an error if a block cannot be expanded, and writes an NGD file containing the unexpanded block.

You may want to run NGDBuild with the –u option to perform preliminary mapping, placement and routing, timing analysis, or simulation on the design even though the design is not complete. To ensure the unexpanded blocks remains in the design when it is mapped, run the MAP program with the –u (Do Not Remove Unused Logic) option.

–uc (User Constraints File)

-uc *ucf_file*[**.ucf**]

The –uc option specifies a UCF (User Constraints File) for the Netlister Launcher to read. The UCF file contains timing and layout constraints that affect how the logical design is implemented in the target device.

The user constraints file must have a .ucf extension. If you specify a user constraints file without an extension, NGDBuild appends the .ucf extension to the file name. If you specify a file name with an extension other than .ucf, you get an error message and NGDBuild does not run.

If you do not enter a –uc option and a UCF file exists with the same base name as the input design file and a .ucf extension, NGDBuild automatically reads the constraints in this UCF file.

The User Constraints File is described in the "User Constraints (UCF) File" chapter.

Note If you use this option, do not use the –i option.

–ur (Read User Rules File)

-ur *rules_file*[**.urf**]

The –ur option specifies a user rules file for the Netlister Launcher to access. This file determines the acceptable netlist input files, the netlist readers that read these files, and the default netlist reader options. This file also allows you to specify third party tool commands for processing designs.

The user rules file must have a .urf extension. If you specify a user rules file with no extension, NGDBuild appends the .urf extension to the file name. If you specify a file name with an extension other than .urf, you get an error message and NGDBuild does not run.

The user rules file is described in the "User Rules File" section of the "EDIF2NGD, XNF2NGD, and NGDBuild" appendix.

Netlister Launcher

The Netlister Launcher, which is part of NGDBuild, performs any netlist translations necessary to execute the NGDBuild command. The Netlister Launcher is described in detail in the "Netlister Launcher" section of the "EDIF2NGD, XNF2NGD, and NGDBuild" appendix.

File Names and Locations

Following are some notes about file names and notations in NGDBuild.

- An intermediate file has the same root name as the design that produced it. An intermediate file is generated when more than one netlist reader is needed to translate a netlist to a NGO file.
- Netlist root file names in the search path must be unique. For example, if you have the design state.edn, you cannot have another design named state.xnf in any of the directories specified in the search path.

- NGDBuild and the Netlister Launcher support quoted file names. Quoted file names may have special characters (for example, a space) that are not normally allowed.
- If the output directory specified in the call to NGDBuild is not writable, an error is displayed and NGDBuild fails.

Chapter 5

User Constraints (UCF) File

This chapter contains the following sections.

- "Overview"
- "UCF Flow"

Overview

The UCF file is an ASCII file specifying constraints on the logical design. You create this file and enter your constraints in the file with a text editor.

Note You can also use the Constraints Editor to create constraints within a UCF file. Refer to the *Constraints Editor Guide* for details.

These constraints affect how the logical design is implemented in the target device. The file can also be used to override constraints specified during design entry.

The following types of logical constraints can be included in the UCF file.

- Placement
- Mapping
- Timing
- BitGen

These constraints and the syntax for entering them in the UCF are described in the "Attributes, Constraints, and Carry Logic" chapter of the *Libraries Guide*. A more detailed description of timing constraints can be found in the "Using Timing Constraints" chapter of this manual, the *Development System Reference Guide*.

UCF Flow

The following figure illustrates the UCF flow.

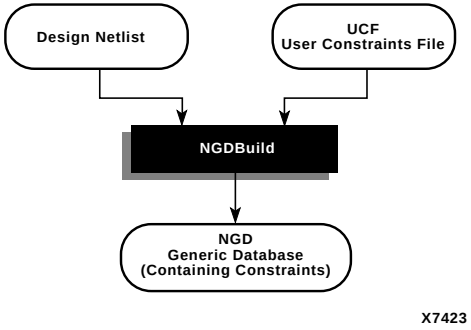


Figure 5-1 UCF File Flow

The UCF file is an input to NGDBuild (see the preceding figure). The constraints in the UCF file become part of the information in the NGD file produced by NGDBuild. Some of these constraints are used when the design is mapped by MAP and some of the constraints are written into the PCF (Physical Constraints File) produced by MAP. The constraints in the PCF file are used by the each of the physical design tools (for example, PAR and the timing analysis tools), which are run after your design is mapped.

Chapter 6

Using Timing Constraints

The timing constraints described in this chapter are compatible with the following families.

- XC3000A / L
- XC3100A / L
- XC4000E / L
- XC4000EX / XL / XLA / XV
- XC5200
- Virtex / -E / -II
- Spartan / XL / -II

For information about using timing constraints with CPLDs, consult the *CPLD Schematic Design Guide* and the *CPLD Synthesis Design Guide*.

This chapter describes how you specify timing constraints and contains the following sections.

- "Timing Requirements and Xilinx Software"
- "IOB Register Specification and Reporting"
- "Entering Timing Specifications"
- "Specifying Groups"
- "Defining a Clock Period (PERIOD Constraint)"
- "OFFSET Timing Specifications"
- "Ignoring Selected Paths (TIG)"
- "Basic FROM –TO Syntax"

- "Specifying Timing Points"
- "Using TPTHRU or TPSYNC in a FROM–TO Constraint"
- "Specifying Time Delay in TS Attributes"
- "Using the PRIORITY Keyword"
- "Sample Schematic Using TIMESPEC / TIMEGRP Symbol"
- "Prorating Constraints"
- "Additional Timing Constraints"
- "Constraints Priority"
- "Syntax Summary"

Timing Requirements and Xilinx Software

Xilinx software enables you to specify precise timing requirements for your Xilinx FPGA designs. You can specify the timing requirements for any nets or paths in your design. One way of specifying path requirements is to first identify a set of paths by identifying a group of start and end points. The start and end points can be flip-flops, I / O pads, latches, or RAMs. You can then control the worst-case timing on the set of paths by specifying a single delay requirement for all paths in the set.

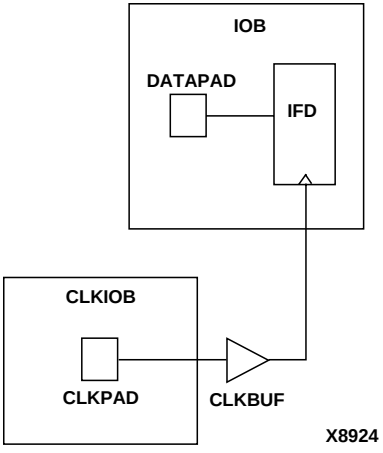
The primary method of specifying timing requirements is by entering them on the schematic. However, you can also specify timing requirements in constraints files (UCF and PCF). For detailed information about the constraints you can use with your schematic entry software, refer to the "Attributes, Constraints, and Carry Logic" chapter of the *Libraries Guide*.

Once you define timing specifications and then map the design, PAR places and routes your design based on these requirements.

To analyze the results of your timing specifications, use TRACE (Timing Report , Circuit Evaluator, and TSI Report). Refer to the "TRACE" chapter for more information.

IOB Register Specification and Reporting

In the 3.1i release, the Xilinx timing tool analyzes OFFSET and FROM TO constraints that include IOB registers. The timing tool reports paths that start or end at IOB registers (including paths between components). This strategy requires the analysis of paths that internally have no length (that is, no components or connections), only a setup requirement, for pad-to-setup paths that originate at a pad and terminate at the input register within the same IOB. In the following example, the pad from the IOB pad to the input register IFD is analyzed and reported by the timing tool.



When these paths are analyzed for either TRACE or PAR, they may create timing errors that cannot be corrected because the timing requirement is less than the setup time for the I/O register. Under these conditions, TRACE always generates a timing error and PAR ceases, indicating that the place and route of the design is impossible due to existing constraints.

Entering Timing Specifications

This section describes the basic methods for entering timing specifications in a schematic or User Constraints File (UCF).

The following notes apply to Mentor Graphics users.

- The term *attribute* in this chapter is equivalent to *property* as used in the Mentor Graphics environment.
- The Mentor netlist writer (ENWRITE) converts all property names to lowercase letters, and the Xilinx netlist reader EDIF2NGD then converts the property names to uppercase letters. Because property names are processed in this way, you must enter variable text in certain constraints in upper case letters only. This requirement is discussed in the following sections.
  - "Entering Timing Specifications in a Schematic"
  - "Creating New Groups from Existing Groups"

Entering Timing Specifications in a Schematic

The TIMESPEC schematic primitive, as illustrated in Figure 6-1, serves as a placeholder for timing specifications, which are called TS attribute definitions. Every TS attribute must be defined in a TIMESPEC primitive, and only TIMESPEC primitives can carry TS attribute definitions. Every TS attribute begins with the letters "TS" and ends with a unique identifier that can consist of letters, numbers, or the underscore character (_).

TS attribute definitions can be any length, but only 30 characters are displayed in the TIMESPEC window. Each TIMESPEC primitive can hold up to eight TS attributes. If you want to include more than eight TS attributes, you can use multiple TIMESPEC primitives in your schematic.

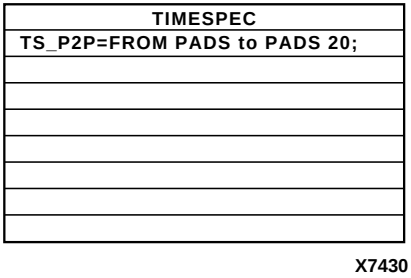


Figure 6-1 TIMESPEC Primitive

How you add a TIMESPEC primitive to your schematic depends on your specific schematic-entry software. Refer to the appropriate Xilinx *Interface Guide* for step-by-step instructions.

A TS attribute defines the allowable delay for paths in your design. The basic syntax for a TS attribute is as follows.

TS*identifier*=**FROM** *source_group* **TO** *dest_group delay*

TS*identifier* is a unique name for the TS attribute, *source_group* and *dest_group* are groups of start points and end points, and *delay* defines the maximum delay for the paths between the start points and end points. The *delay* parameter defines the maximum delay for the attribute. Nanoseconds are the default units for specifying delay time in TS attributes. You can also specify delay using other units, such as picoseconds or megahertz.

Note Keywords, such as FROM, TO, and TS appear in the documentation in upper case; however, you can enter them in the TIMESPEC primitive in either upper or lower case. The characters in the keywords must be all upper case or all lower case. Examples of acceptable keywords are: FROM, TO, from, to. Examples of unacceptable keywords are: From, To, fRoM, tO

Note: The Mentor netlist writer (ENWRITE) converts all property names to lower case letters, and the Xilinx netlist reader EDIF2NGD then converts the property names to upper case letters. To ensure references from one constraint to another are processed correctly, a **TS***identifier* name should contain only upper case letters on a Mentor Schematic (TSMAIN, for example, but not TSmain or TSMain).

Also, if a TSidentifier name is referenced in a property value, it must be entered in upper case letters. For example, the TSID1 in the second constraint below must be entered in upper case letters to match the TSID1 name in the first constraint.

```
TSID1 = FROM gr1 TO gr2 50;


TSMAIN = FROM here TO there TSID1 /2;
```

The basic TS attribute is described in detail in the "Basic FROM –TO Syntax" section. More detailed forms of the attribute are also described in that section.

Note A colon may be used as a separator instead of a space in all timing specifications.

Entering Timing Specifications in a Constraints File

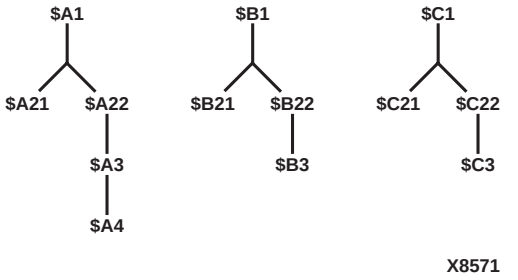
You can enter timing specifications as constraints in a UCF file. When you then run NGDBuild on your design, your timing specifications are added to the design database as part of the NGD file.

To avoid manually entering timing constraints in a UCF file, use the Xilinx Constraints Editor, a tool that greatly simplifies constraint creation. For a detailed description of how to use the editor, see the *Constraints Editor Guide*.

The basic syntax for a timing specification entered in a constraints file is the TS attribute syntax described in the "Basic FROM –TO Syntax" section.

Although not required, Xilinx recommends that NET and INST names be enclosed in double quotes to avoid errors. Additionally, inverted signal names that contain a tilde, for example, ~OUTSIG1, must always be enclosed in double quotes. Other special characters that must be enclosed in quotes are the asterisk (*) and question mark (?).

You can use the wildcard character (*) to traverse the hierarchy of a directory within a UCF or NCF file. Consider the following directory hierarchy.



With the example hierarchy, the following specifications illustrate the scope of the wildcard.

```
   INST *          => <everything>

INST /*          => <everything>
INST /*/         => <$A1,$B1,$C1>
INST $A1/*       => <$A21,$A22,$A3,$A4>
INST $A1/*/      => <$A21,$A22>
INST $A1/*/*     => <$A3,$A4>
INST $A1/*/*/    => <$A3>
INST $A1/*/*/*   => <$A4>
INST $A1/*/*/*/  => <$A4>
INST /*/*22/     => <$A22,$B22,$C22>
INST /*/*22      => <$A22,$A3,$A4,$B22,$B3,$C22,$C3>
INST /*/*22/*    => <$A3,$A4,$B3,$C22,$C3>
```

Specifying Groups

In a TS attribute, you specify the set of paths to be analyzed by grouping start and end points in one of the following ways.

- Refer to a predefined group by specifying one of the corresponding keywords — FFS, PADS, LATCHES, or RAMS.
- Create your own groups within a predefined group by tagging symbols with TNM (pronounced tee-name) attributes.
- Create groups that are combinations of existing groups using TIMEGRP symbols.
- Create groups by pattern matching on net names.

The following sections discuss each method in detail.

Using Predefined Groups

You can refer to a group of flip-flops, input latches, pads, or RAMs by using the corresponding keywords.

Keyword	Description
FFS	CLB or IOB flip-flops only; not flip-flops built from function generators (Shift Register LUTs are included in Virtex/-E/-II and Spartan-II devices also)
LATCHES	CLB or IOB latches only; not latches built from function generators
PADS	Input/Output pads
RAMS	For architectures with RAMS (LUT RAMS and Block RAMS are included for Virtex/-E/-II and Spartan-II devices but Shift Register LUTs are not included in RAMS)

From-To statements enable you to define timing specifications for paths between predefined groups. The following examples are TS attributes that reside in the TIMESPEC primitive or are entered in the UCF. This method enables you to easily define default timing specifications for the design, as illustrated by the following examples.

Schematic syntax in TIMESPEC primitive

```
TS01=FROM FFS TO FFS 30
TS02=FROM LATCHES TO LATCHES 25
TS03=FROM PADS TO RAMS 70
TS04=FROM FFS TO PADS 55
```

UCF syntax

```
TIMESPEC TS01=FROM FFS TO FFS 30;
TIMESPEC TS02=FROM LATCHES TO LATCHES 25;
TIMESPEC TS03=FROM PADS TO RAMS 70;
TIMESPEC TS04=FROM FFS TO PADS 55;
```

A predefined group can also carry a name qualifier; the qualifier can appear any place where the predefined group is used. This name qualifier restricts the number of elements being referred to. The syntax used is as follows.

predefined group **(***name_qualifier* [*name_qualifier*]**)**

name_qualifier is the full hierarchical name of the net that is sourced by the primitive being identified.

The name qualifier can include wildcard characters (*) to indicate any number of characters (or ? to indicate a single character) which allows the specification of more than one net or allows you to shorten the full hierarchical name to something that is easier to type.

As an example, specifying the group FFS(MACRO_A/Q?) selects only the flip-flops driving the Q0, Q1, Q2 and Q3 nets in the following macro.

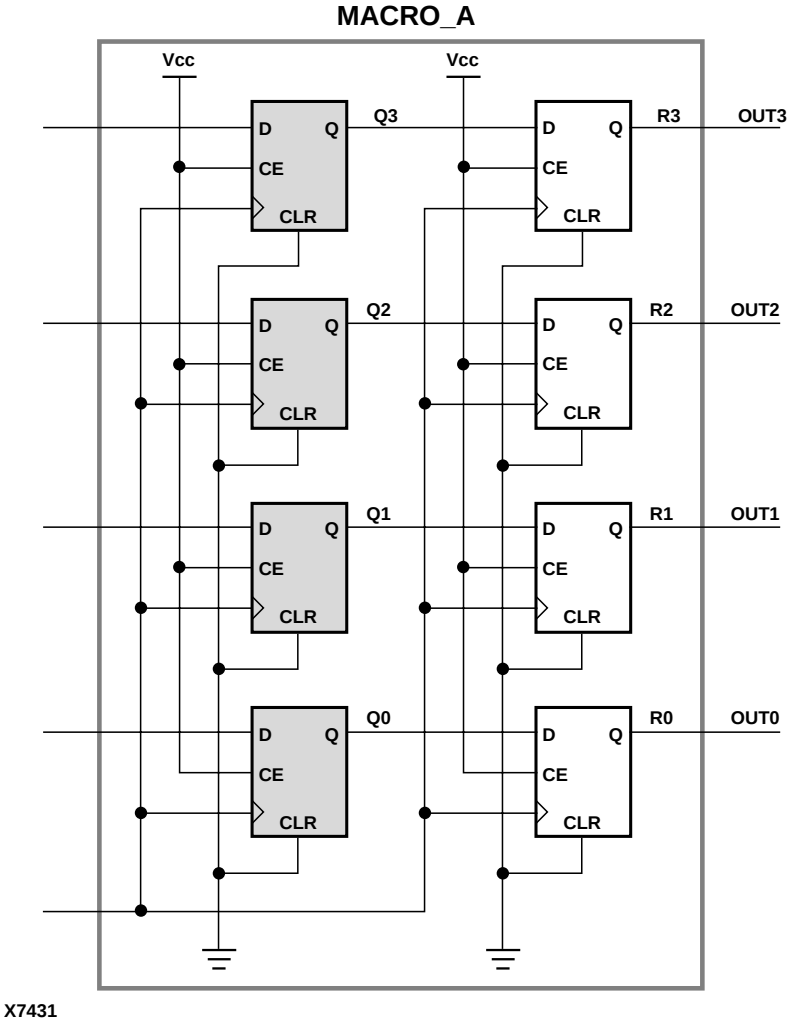


Figure 6-2 Using Qualifiers with Predefined Groups

To create more specific groups see the following section.

Creating User-Defined Groups Using TNMs

A TNM (timing name) is an attribute that can be used to identify the elements that make up a group which can then be used in a timing specification. A TNM is a property that you place directly on your schematic to tag a specific net, element pin, primitive or macro. All symbols tagged with the TNM identifier are considered a group. Place TNM attributes directly on your schematic or in a UCF file using the following syntax. Special rules apply when using TNM with the PERIOD constraint for Virtex/-E/-II and Spartan-II CLKDLLs. Refer to the "PERIOD Specifications on CLKDLLs" section.

Schematic syntax

TNM=*identifier*

UCF syntax

{**NET** | **INST** | **PIN**} *object_name* **TNM=***identifier*;

identifier is a value that consists of any combination of letters, numbers, or underscores. Keep the TNM short for convenience and clarity.

Do not use the reserved words FFS, LATCHES, PADS, RAMS, RISING, FALLING, TRANSHI, TRANSLO, or EXCEPT, as *identifiers*. The constraints in the table below are also reserved words and should not be used as *identifiers*.

Reserved Words (Constraints)		
ADD	FAST	NODELAY
ALU	FBKINV	OPT
ASSIGN	FILE	OSC
BEL	F_SET	RES
BLKNM	HBLKNM	RLOC
CAP	HU_SET	RLOC_ORIGIN
CLKDV_DIVIDE	H_SET	RLOC_RANGE
CLBNM	INIT	SCHNM
CMOS	INIT OX	SLOW
CYMODE	INTERNAL	STARTUP_WAIT

Reserved Words (Constraints)		
DECODE	IOB	SYSTEM
DEF	IOSTANDARD	TNM
DIVIDE1_BY	LIBVER	TRIM
DIVIDE2_BY	LOC	TS
DOUBLE	LOWPWR	TTL
DRIVE	MAP	TYPE
DUTY_CYCLE_ CORRECTION	MEDFAST	USE_RLOC
EQN	MEDSLOW	U_SET
FAST	MINIM	

Note: If you want to use a keyword as an identifier, you can enclose the keyword in quotation marks. In the TNM statement **TNM=RAMS ”CMOS”**, CMOS is treated as an identifier instead of a keyword.

You can specify as many groups of end points as are necessary to describe the performance requirements of your design. However, to simplify the specification process and reduce the place and route time, use as few groups as possible.

A predefined group can be used in a TNM specification, using the following syntax on a schematic or UCF file.

Schematic syntax

TNM=*predefined_group identifier*

UCF syntax

{**NET** | **INST** | **PIN**} *object_name* **TNM=***predefined_group identifier*;

The *object_name* is the net, pin, or instance name.

The *predefined_group* is one of the groups (for example, FFS or RAMS) defined in the “Using Predefined Groups” section and *identifier* is a value that consists of any combination of letters, numbers, or underscores. Paths defined by the TNM are traced forward if placed on a net or pin, through any number of gates or buffers, until they reach a member of the *predefined_group*. That element is added to the speci-

fied TNM group. TNM does not trace through the element to the next element; forward tracing stops at the element.

This mechanism is called forward tracing. If TNM is placed on an instance, paths are traced "downward" through a hierarchy instead of forward along a net.

Note If a TNM is placed on an input pad net, the constraint only applies to the input pad. In that case, refer to the "Creating User-Defined Groups Using TNM_NET" section.

The specification shown below, when attached to a net, would create a group called FIFO_CORE consisting of all of the RAM primitives traced forward on the net. The specification shows the schematic and UCF syntax.

Schematic syntax

```
TNM=RAMS FIFO_CORE
```

UCF syntax

```
NET net_name TNM=RAMS FIFO_CORE;
```

The following figure illustrates the preceding TNM identifier. The two RAMs traced forward from the net are included in the group. The flip flop is not.

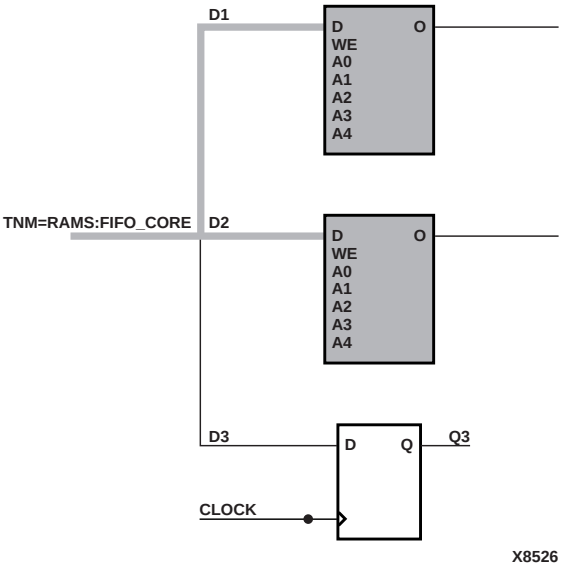


Figure 6-3 TNM Placed on a Net

A defined net in a TNM statement can have a name qualifier (for example, TNM=FFS (FRED*) GRP_A), as described in the "Creating Groups by Pattern Matching" section.

You can use several methods for tagging groups of end points: placing identifiers on nets, macro or primitive pins, primitives, or macro symbols. Which method you choose depends on how the path end points are related in your design. For each of these elements, you can use the predefined group syntax described earlier in this section.

The following subsections discuss the different methods of placing TNMs in your design. The same TNM attribute can be used as many ways and as many times as necessary to get the TNM applied to all of the elements in the desired group.

You can place TNM attributes in either of two places: in the schematic as discussed in this section or in a constraints file (UCF or NCF).

The syntax for specifying TNMs in a UCF or NCF constraints file is described in the "Attributes, Constraints, and Carry Logic" chapter of the *Libraries Guide*.

Placing TNMs on Nets

The TNM attribute can be placed on any net in the design. The attribute indicates that the TNM value should be attached to all valid elements fed by all paths that fan forward from the tagged net. Forward tracing stops at any flip-flop, latch, RAM or pad. See Figure 6-3. TNMs do not propagate across IBUFs if they are attached to the input pad net. Also refer to the “Creating User-Defined Groups Using TNMs” section.

Placing TNMs on Macro or Primitive Pins

The TNM attribute can be placed on any macro or component pin in the design if the design entry package allows placement of attributes on macro or primitive pins. The attribute indicates that the TNM value should be attached to all valid elements fed by all paths that fan forward from the tagged pin. Forward tracing stops at any flip-flop, latch, RAM or pad. The following illustration shows the valid elements for a TNM attached to the schematic a macro pin.

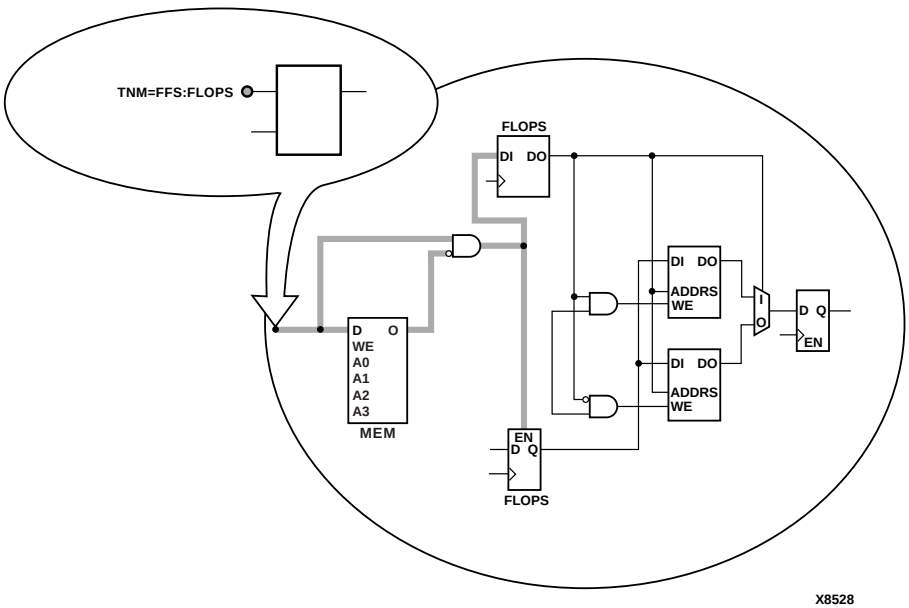


Figure 6-4 TNM Placed on a Macro Pin

The syntax for the UCF file would be

```
PIN pin_name TNM=FFS FLOPS;
```

Placing TNMs on Primitive Symbols

You can group individual logic primitives explicitly by flagging each symbol, as illustrated by the following figure.

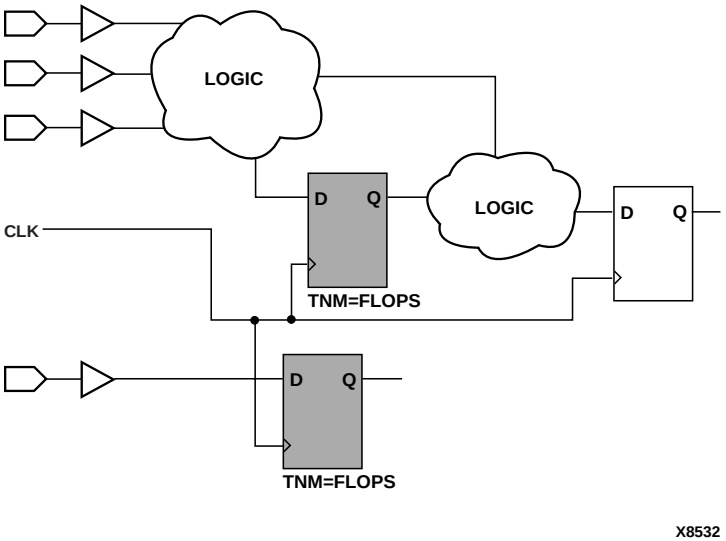


Figure 6-5 TNM on Primitive Symbols

In the figure, the flip-flops tagged with the TNM form a group called "'FLOPS." The untagged flip-flop on the right side of the drawing is not part of the group.

Place only one TNM on each symbol, driver pin, or macro driver pin.

Schematic syntax

```
TNM=FLOPS
```

UCF syntax

```
INST symbol_name TNM=FLOPS;
```

Placing TNMs on Macro Symbols

A macro is an element that performs some general purpose higher level function. It typically has a lower level design that consists of primitives, other macros, or both, connected together to implement

the higher level function. An example of a macro function is a 16-bit counter.

A TNM attribute attached to a macro indicates that all elements inside the macro (at all levels of hierarchy below the tagged macro) are part of the named group.

When a macro contains more than one symbol type and you want to group only a single type, use the TNM identifier in conjunction with one of the predefined groups: FFS, RAMS, PADS, or LATCHES as indicated by the following syntax examples.

Schematic syntax

```
TNM=FFS identifier
TNM=RAMS identifier
TNM=LATCHES identifier
TNM=PADS identifier
```

UCF syntax

```
INST macro_name TNM=FFS identifier;
INST macro_name TNM=RAMS identifier;
INST macro_name TNM=LATCHES identifier;
INST macro_name TNM=PADS identifier;
```

If multiple symbols of the same type are contained in the same hierarchical block, you can simply flag that hierarchical symbol, as illustrated by the following figure. In the figure, all flip-flops included in the macro are tagged with the TNM “FLOPS”. By tagging the macro symbol, you do not have to tag each underlying symbol individually.

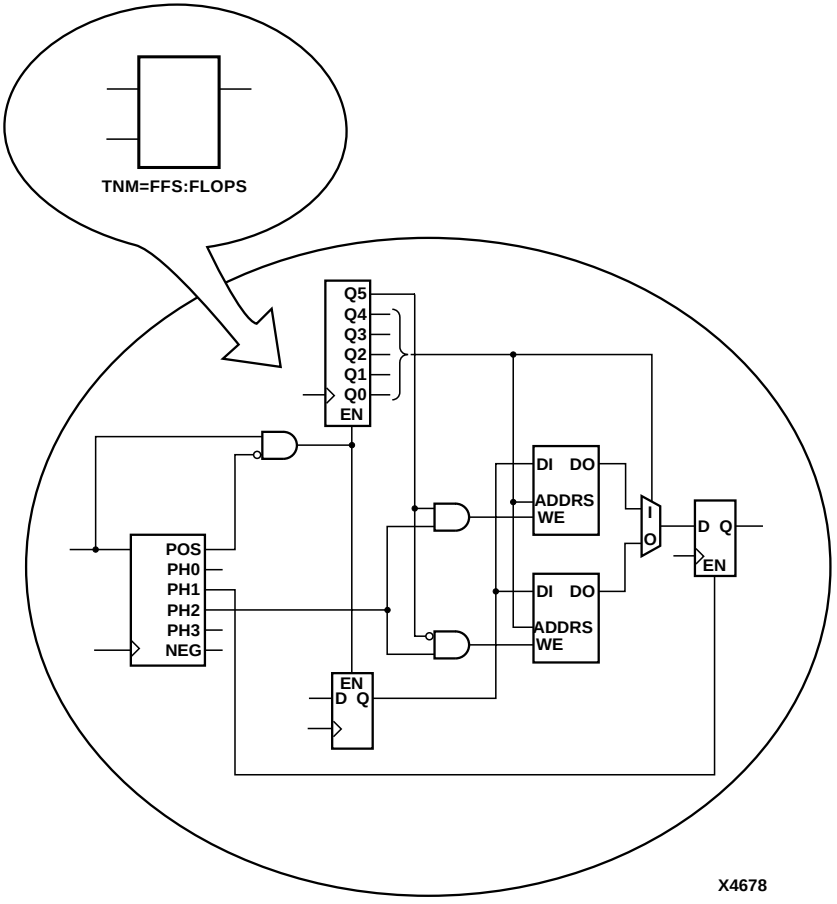


Figure 6-6 TNM on Macro Symbol

Placing TNMs on Nets or Pins to Group Flip-Flops and Latches

You can easily group flip-flops, latches, or both by flagging a common input net, typically either a clock net or an enable net. If you attach a TNM to a net or driver pin, that TNM applies to all flip-flops and input latches that are reached through the net or pin. That is, that path is traced forward, through any number of gates or buffers, until it reaches a flip-flop or input latch. That element is added to the specified TNM group.

The following figure illustrates the use of a TNM on a net that traces forward to create a group of flip-flops.

In the figure, the attribute TNM=FLOPS traces forward to the first two flip-flops, which form a group called FLOPS. The bottom flip-flop is not part of the group FLOPS.

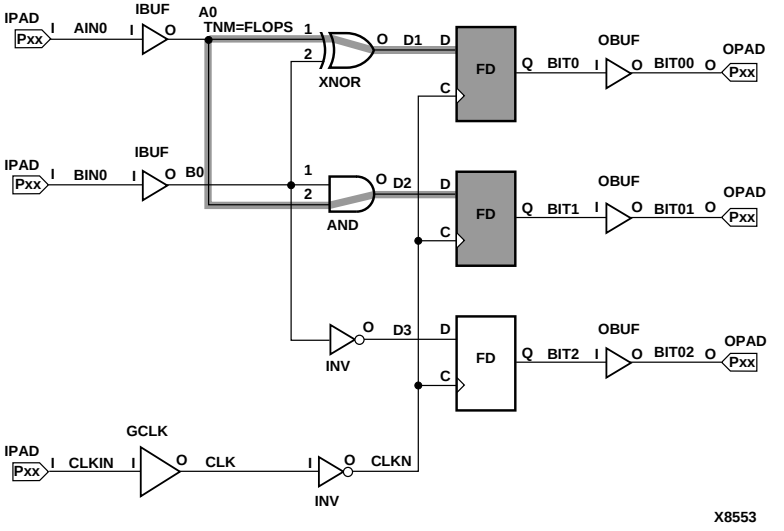


Figure 6-7 TNM on Net Used to Group Flip-Flops

The following figure illustrates placing a TNM on a clock net, which traces forward to all three flip-flops and forms the group Q_FLOPS.

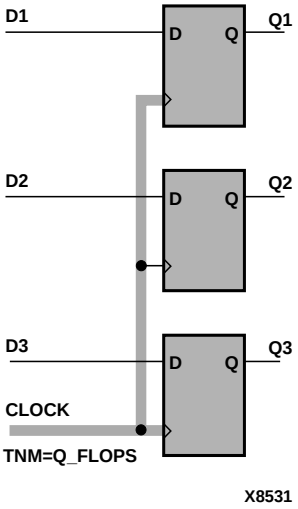


Figure 6-8 TNM on Clock Pin Used to Group Flip-Flops

The TNM parameter on nets or pins is allowed to have a qualifier.

For example, on schematics

TNM=FFS data

TNM=RAMS fifo

TNM=LATCHES capture

In UCF files

{**NET** | **PIN**} *net_or_pin_name* **TNM=FFS data;**

{**NET** | **PIN**} *net_or_pin_name* **TNM=RAMS fifo;**

{**NET** | **PIN**} *net_or_pin_name* **TNM=LATCHES capture;**

A qualified TNM is traced forward until it reaches the first storage element (flip-flop, latch, or RAM). If that type of storage element matches the qualifier, the storage element is given that TNM value. Whether or not there is a match, the TNM is *not* traced through that storage element.

TNM parameters on nets or pins are never traced through a storage element (flip-flop, latch or RAM).

Creating User-Defined Groups Using TNM_NET

A TNM_NET (timing name for nets) is an attribute that can be used to identify the elements that make up a group which can then be used in a timing specification. Essentially TNM_NET is equivalent to TNM on a net *except* for pad nets. Special rules apply when using TNM_NET with the PERIOD constraint for Virtex/-E/-II CLKDLLs. Refer to the "PERIOD Specifications on CLKDLLs" section.

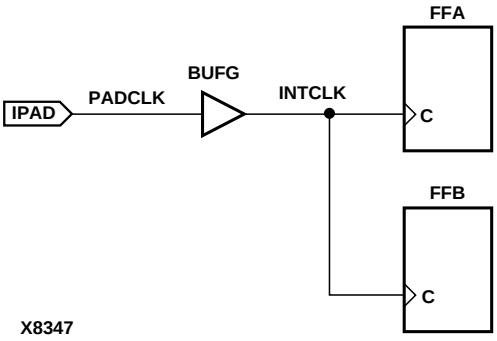
A TNM_NET is a property that you normally use in conjunction with an HDL design to tag a specific net. All nets tagged with the TNM_NET identifier are considered a group. The UCF syntax is as follows.

```
NET net_name TNM_NET=identifier;
```

identifier is a value that consists of any combination of letters, numbers, or underscores. Keep the TNM_NET short for convenience and clarity. The basic syntax rules for TNM_NET and TNM are identical. Refer to the "Creating User-Defined Groups Using TNM_NET" section for details.

The TNM_NET attribute can be used to define certain types of nets that cannot be adequately described by the TNM constraint. This attribute is specifically targeted for use in HDL designs.

For example, consider the following design.



In the preceding design, a TNM associated with the PAD symbol only includes the PAD symbol as a member in a timing analysis group. For example, the following UCF file entry creates a time group that includes the IPAD symbol only.

```
NET PADCLK TNM=PADS(*) PADGRP; (UCF file example)
```

However, using TNM to define a time group for the net PADCLK creates an empty time group.

```
NET PADCLK TNM=FFS(*) FFGRP;(UCF file example)
```

All properties that apply to a pad are transferred from the net to the PAD symbol. Since the TNM is transferred from the net to the PAD symbol, the qualifier, “FFS(*)” does not match the PAD symbol.

To overcome this obstacle for schematic designs using TNM, you can create a time group for the INTCLK net.

```
NET INTCLK TNM=FFS(*) FFGRP;(UCF file example)
```

However, for HDL designs, the only meaningful net names are the ones connected directly to pads. Then, use TNM_NET to create the FFGRP time group.

```
NET PADCLK TNM_NET=FFS(*) FFGRP;(UCF file example)
```

NGDBuild does not transfer a TNM_NET attribute from a net to a PAD as it does with TNM.

TNM_NET can be used in NCF or UCF files as a property attached to an object in an input netlist (EDIF or XNF). TNM_NET is not supported in PCF files.

TNM_NET can only be used with nets. If TNM_NET is used with any other object such as a pin or symbol, a warning is generated and the TNM_NET definition is ignored.

Creating New Groups from Existing Groups

In addition to naming groups using the TNM identifier, you can also define groups in terms of other groups. You can create a group that is a combination of existing groups by defining a TIMEGRP attribute as follows.

Schematic syntax in TIMEGRP primitive

```
newgroup=existing_grp1 existing_grp2 [ existing_grp3
. . .]
```

UCF syntax

```
TIMEGRP newgroup=existing_grp1 existing_grp2[
existing_grp3 ...];
```

newgroup is a newly created group that consists of existing groups created via TNMs, predefined groups, or other TIMEGRP attributes.

The Mentor netlist writer (ENWRITE™) converts all property names to lower case letters, and the Xilinx netlist reader EDIF2NGD then converts the property names to upper case letters. To ensure references from one constraint to another are processed correctly,

- Group names should contain only upper case letters on a Mentor Schematic (MY_FLOPS, for example, but not my_flops or My_flops).
- If a group name appears in a property value, it must also be expressed in upper case letters. For example, the GROUP3 in the first constraint below must be entered in upper case letters to match the GROUP3 in the second constraint.

  Schematic syntax in TIMEGRP primitive

  ```
  GROUP1 = gr2 GROUP3
  GROUP3 = FFS except grp5
  ```

  UCF syntax

  ```
  TIMEGRP GROUP1 = gr2 GROUP3;
  TIMEGRP GROUP3 = FFS except grp5;
  ```

TIMEGRP attributes reside in the TIMEGRP primitive, as illustrated in the figure below. Once you create a TIMEGRP attribute definition within a TIMEGRP primitive, you can use it in the TIMESPEC primitive. Each TIMEGRP primitive can hold up to eight group definitions. Since your design might include more than eight TIMEGRP attributes, you can use multiple TIMEGRP primitives.

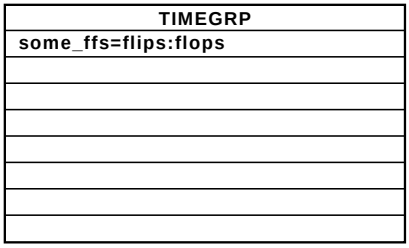


Figure 6-9 TIMEGRP Primitive

You can place TIMEGRP attributes in either of two places: in the TIMEGRP primitive on the schematic as discussed in this section or in a constraints file (UCF or NCF). The syntax for specifying TIMEGRPs in a UCF or NCF constraints file is described in the "Attributes, Constraints, and Carry Logic" chapter of the *Libraries Guide*.

You can use TIMEGRP attributes to create groups using the following methods.

- Combining multiple groups into one
- Creating groups by exclusion
- Defining flip-flop subgroups by clock sense

The following subsections discuss each method in detail.

Combining Multiple Groups into One

You can define a group by combining other groups. The following syntax example illustrates the simple combining of two groups.

Schematic syntax in TIMEGRP primitive

```
big_group=small_group medium_group
```

UCF syntax

```
TIMEGRP big_group=small_group medium_group;
```

In this syntax example, small_group and medium_group are existing groups defined using a TNM or TIMEGRP attribute. Within the TIMEGRP primitive, TIMEGRP attributes can be listed in any order; that is, you can create a TIMEGRP attribute that references another TIMEGRP attribute that appears after the initial definition.

Note A circular definition, as shown below, causes an error when you run your design through NGDBuild.

Schematic syntax in TIMEGRP primitive

```
many_ffs=ffs1 ffs2
ffs1=many_ffs ffs3
```

UCF syntax

```
TIMEGRP many_ffs=ffs1 ffs2;
TIMEGRP ffs1=many_ffs ffs3;
```

Creating Groups by Exclusion

You can define a group that includes all elements of one group except the elements that belong to another group, as illustrated by the following syntax examples.

Schematic syntax in TIMEGRP primitive

group1=*group2* **EXCEPT** *group3*

UCF syntax

TIMEGRP *group1*=*group2* **EXCEPT** *group3*;

- *group1* represents the group being defined. It contains all of the elements in *group2* except those that are also in *group3*.
- *group2* and *group3* can be a valid TNM, predefined group, or TIMEGRP attribute.

As illustrated by the following example, you can specify multiple groups to include or exclude when creating the new group.

Schematic syntax in TIMEGRP primitive

group1=*group2 group3* **EXCEPT** *group3 group5*

UCF syntax

TIMEGRP *group1*=*group2 group3* **EXCEPT** *group4 group5*;

The example defines a *group1* that includes the members of *group2* and *group3*, except for those members that are part of *group4* or *group5*. All of the groups before the keyword EXCEPT are included, and all of the groups after the keyword are excluded.

Certain reserved words cannot be used as group names. These reserved words are described in the "Creating User-Defined Groups Using TNM_NET" section

Defining Flip-Flop Subgroups by Clock Sense

You can create subgroups using the keywords RISING and FALLING to group flip-flops triggered by rising and falling edges.

Schematic syntax in TIMEGRP primitive

```
group1=RISING ffs
group2=RISING ffs_group
group3=FALLING ffs
group4=FALLING ffs_group
```

UCF syntax

```
TIMEGRP group1=RISING ffs;

TIMEGRP group2=RISING ffs_group;
TIMEGRP group3=FALLING ffs;
TIMEGRP group4=FALLING ffs_group;
```

group1 to *group4* are new groups being defined. The *ffs_group* must be a group that includes only flip-flops.

Note Keywords, such as EXCEPT, RISING, and FALLING, appear in the documentation in upper case; however, you can enter them in the TIMEGRP primitive in either lower or upper case. You cannot enter them in a combination of lower and upper case.

The following example defines a group of flip-flops that switch on the falling edge of the clock.

Schematic syntax in TIMEGRP primitive

```
falling_ffs=FALLING ffs
```

UCF syntax

```
TIMEGRP falling_ffs=FALLING ffs;
```

Defining Latch Subgroups by Gate Sense

Groups of type LATCHES (no matter how these groups are defined) can be easily separated into transparent high and transparent low subgroups. The TRANSHI and TRANSLO keywords are provided for this purpose, and are used in TIMEGRP statements like the RISING and FALLING keywords for flip-flop groups. For example

Schematic syntax in TIMEGRP primitive

```
lowgroup=TRANSLO latchgroup highgroup=TRANSHI
latchgroup
```

UCF syntax

```
TIMEGRP lowgroup=TRANSLO latchgroup;
TIMEGRP highgroup=TRANSHI latchgroup;
```

Creating Groups by Pattern Matching

When creating groups, you can use wildcard characters to define groups of symbols whose associated net names match a specific pattern.

How to Use Wildcards to Specify Net Names

The wildcard characters, * and ?, enable you to select a group of symbols whose output net names match a specific string or pattern.

The asterisk (*) represents any string of zero or more characters. The question mark (?) indicates a single character.

For example, DATA* indicates any net name that begins with "DATA," such as DATA, DATA1, DATA22, DATABASE, and so on. The string NUMBER? specifies any net names that begin with "NUMBER" and end with one single character, for example, NUMBER1, NUMBERS but not NUMBER or NUMBER12.

You can also specify more than one wildcard character. For example, *AT? specifies any net names that begin with any series of characters followed by "AT" and end with any one character such as BAT1, CAT2, and THAT5. If you specify *AT??, you would match BAT11, CAT26, and THAT50.

Pattern Matching Syntax

The syntax for creating a group using pattern matching is shown below.

Schematic syntax in TIMEGRP primitive

```
group=predefined_group(pattern)
```

UCF syntax

```
TIMEGRP group=predefined_group(pattern);
```

predefined_group can only be one of the following predefined groups—FFS, LATCHES, PADS, or RAMS. The *pattern* is any string of characters used in conjunction with one or more wildcard characters.

Note When specifying a net name, you must use its full hierarchical path name so PAR can find the net in the flattened design.

For flip-flops, input latches, and RAMs, specify the output net name. For pads, specify the external net name.

The following example illustrates creating a group that includes the flip-flops that source nets whose names begin with $1I3/FRED.

Schematic syntax in TIMEGRP primitive

```
group1=ffs($1I3/FRED*)
```

UCF syntax

```
TIMEGRP group1=ffs($1I3/FRED*);
```

The following example illustrates a group that excludes certain flip-flops whose output net names match the specified pattern.

Schematic syntax in TIMEGRP primitive

```
this_group=ffs EXCEPT ffs(a*)
```

UCF syntax

```
TIMEGRP this_group=ffs EXCEPT ffs(a*);
```

In this example, `this_group` includes all flip-flops except those whose output net names begin with the letter "a."

The following defines a group named "some_latches".

Schematic syntax in TIMEGRP primitive

```
some_latches=latches($1I3/xyz*)
```

UCF syntax

```
TIMEGRP some_latches=latches($113/xyz*);
```

The group `some_latches` contains all input latches whose output net names start with "$1I3/xyz."

Additional Pattern Matching Details

In addition to using pattern matching when you create timing groups, you can specify a predefined group qualified by a pattern any place you specify a predefined group. The syntax below illustrates how pattern matching can be used within a timing specification.

Schematic syntax in TIMESPEC primitive

TS*identifier***=FROM** *predefined_group***(***pattern***) TO** *predefined_group*
(*pattern***)** *delay*

UCF syntax

TIMESPEC TS*identifier***=FROM**
*predefined_group***(***pattern***) TO** *predefined_group*
(*pattern***)** *delay*;

Instead of specifying just one pattern, you can also specify a list of patterns separated by a colon (:) as illustrated below.

Schematic syntax in TIMEGRP primitive

```
some_ffs=ffs(a*:b?:c*d)
```

UCF syntax

```
TIMEGRP some_ffs=ffs(a*:b?:c*d);
```

The group `some_ffs` contains flip-flops whose output net names adhere to one of the following rules.

- Start with the letter “a”
- Contain two characters; the first character is “b”
- Start with “c” and end with “d”

Defining a Clock Period (PERIOD Constraint)

A clock period specification checks timing clocked by the net (all paths that terminate at a register clocked by the specified net).

The period specification is attached to the clock net. The definition of a clock period is unlike a FROM-TO style specification because the timing analysis tools automatically take into account any inversions of the clock net at register clock pins.

A PERIOD constraint on the clock net in the following figure would generate a check for delays on all paths that terminate at a pin that has a setup or hold timing constraint relative to the clock net. This could include the data paths D1 to CLB1.D, CLB1.Q to CLB2.D, as well as the path EN to CLB2.EC (if the enable were synchronous with respect to the clock).

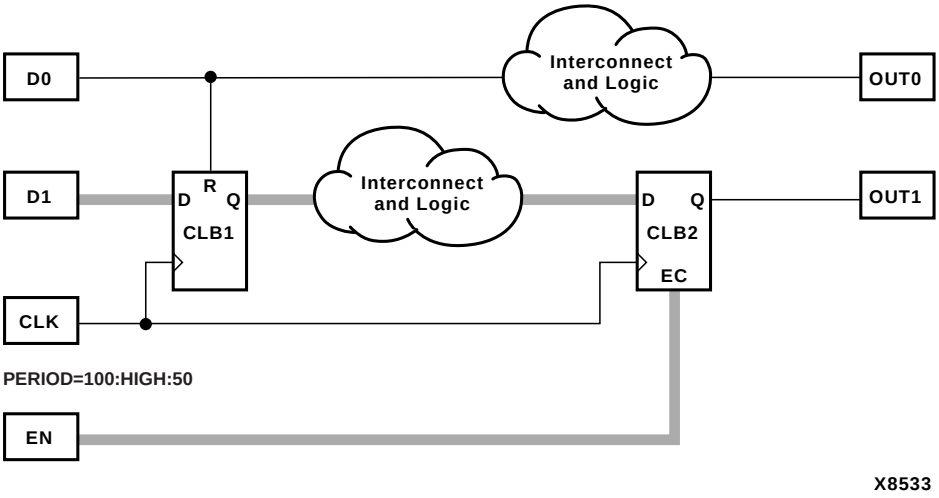


Figure 6-10 Paths for PERIOD Constraint

In 3.1i, the timing tool no longer checks pad-to-register paths relative to setup requirements. For example in the preceding figure, the path from D1 to Pin D of CLB1 is no longer included in the PERIOD constraint.

Simple Method

A simple method of defining a clock period is to attach the following attribute directly to a net in the path that drives the register clock pins.

Schematic syntax

PERIOD = *period* { **HIGH** | **LOW** } [*high_or_low_time*]

UCF syntax

[*period_item*] **PERIOD =** *period* { **HIGH** | **LOW** } [*high_or_low_time*] **;**

period_item is one of the following,

- NET "*net_name*"
- TIMEGRP "*group_name*"
- ALLCLOCKNETS (PCF only)

period is the required clock period. The default units are nanoseconds, but the timing number can be followed by ps, ns, us, or ms. Units may be entered with or without a leading space, and are case-insensitive. The **HIGH**|**LOW** keyword indicates whether the first pulse in the period is high or low, and the optional *high_or_low_time* is the duty cycle of the first pulse. If an actual time is specified, it must be less than the period. If no high or low time is specified the default duty cycle is 50%. The default units for *high_or_low_time* is ns, but the number can be followed by % or by ps, ns, us or ms if you want to specify an actual time measurement.

The PERIOD constraint is forward traced in exactly the same way a TNM would be and attaches itself to all of the flip-flops that the forward tracing reaches. If a more complex form of tracing behavior is required (for example, where gated clocks are used in the design), you must place the PERIOD on a particular net or use the preferred method described next.

Preferred Method

The preferred method for defining a clock period allows more complex derivative relationships to be defined as well as a simple clock period. The following attribute is attached to a TIMESPEC symbol in conjunction with a TNM attribute attached to the relevant clock net.

Schematic syntax in a TIMESPEC symbol

TS*identifier***=PERIOD** *TNM_reference period* {**HIGH** | **LOW**} [*high_or_low_time*]

UCF syntax

TIMESPEC TS*identifier***=PERIOD** *TNM_reference period* {**HIGH** | **LOW**} [*high_or_low_time*] **;**

identifier is a reference identifier that has a unique name.

TNM_reference is the identifier name that is attached to a clock net (or a net in the clock path) using a TNM attribute.

- NET "*net_name*"
- TIMEGRP "*group_name*"
- ALLCLOCKNETS

The variable name *period* is the required clock period. The default units for *period* are nanoseconds, but the number can be followed by ps, ns, us, or ms. Units may be entered with or without a leading space, and are case-insensitive. The **HIGH**|**LOW** keyword indicates whether the first pulse in the period is high or low, and the optional *high_or_low_time* is the polarity of the first pulse. If an actual time is specified, it must be less than the period. If no high or low time is specified the default duty cycle is 50%. The default units for *high_or_low_time* is ns, but the number can be followed by % or by ps, ns, us, or ms if you want to specify an actual time measurement.

Example:

Clock net sys_clk has the attribute `tnm=master_clk` attached to it and the following attribute is attached to a TIMESPEC primitive.

Schematic syntax in a TIMESPEC symbol

```
TS_master=PERIOD master_clk 50 HIGH 30
```

UCF syntax

```
TIMESPEC TS_master=PERIOD master_clk 50 HIGH 30;
```

This period constraint applies to the net master_clk, and defines a clock period of 50 nanoseconds, with an initial 30 nanosecond high time.

Specifying Derived Clocks

The preferred method of defining a clock period uses an identifier, allowing another clock period specification to reference it. To define the relationship in the case of a derived clock, use the following syntax.

Schematic syntax in a TIMESPEC symbol

```
TSidentifier=PERIOD TNM_reference
another_PERIOD_identifier
[{/|*}number] [{HIGH|LOW} high_or_low_time]
```

UCF syntax

```
TIMESPEC TSidentifier=PERIOD TNM_reference
another_PERIOD_identifier
[{/|*}number] [{HIGH|LOW} high_or_low_time] ;
```

- *identifier* is a reference identifier that has a unique name.
- *TNM_reference* is the identifier name that is attached to a clock net or a net in the clock path using a TNM attribute.
- *another_PERIOD_identifier* is the name of the identifier used on another period specification.
- *number* is a floating point number.
- The **HIGH** | **LOW** keyword indicates whether the first pulse in the period is high or low, and the optional *high_or_low_time* is the polarity of the first pulse. If an actual time is specified it must be less than the period. If no high or low time is specified, the default duty cycle is 50%. The default units for *high_or_low_time* is ns, but the number can be followed by % or by ps, ns, us, or ms if you want to specify an actual time measurement.

Example

A clock net has the attribute tnm=slave_clk attached to it and the following attribute is attached to a TIMESPEC primitive.

Schematic syntax in a TIMESPEC symbol

```
ts_slave1=PERIOD slave_clk TS_master *4
```

UCF syntax

```
TIMESPEC ts_slave1=PERIOD slave_clk TS_master
*4;
```

PERIOD Specifications on CLKDLLs

A TNM or TNM_NET group on a net is usually traced forward to tag all of the flip-flops, latches, and RAMs driven by that net. However, if the group traces into the clock input pin of a Virtex/-E/-II or Spartan-II CLKDLL or CLKDLLHF symbol, it is *not* simply traced through the CLKDLL outputs.

When a TNM or TNM_NET group is traced into the CLKIN pin of a Spartan-II, Virtex/-E/-II CLKDLL component, NGDBuild examines the TNM group to ensure that it meets all the following conditions.

- Used in only *one* PERIOD specification
- Not used in a FROM-TO or OFFSET specification
- Not referenced in any user group definition

Note If a group is traced into the CLKDLL but is not used in exactly one PERIOD specification, NGDBuild issues an error. To tag the elements driven by the CLKDLL, place TNM or TNM_NET groups directly on the CLKDLL output nets.

If the TNM or TNM_NET group meets these conditions, NGDBuild copies the PERIOD specification to each CLKDLL output net and adjusts it as shown in the following table

Output Pin	Adjustments
CLK0 CLK90 CLK180 CLK270	If the DUTY_CYCLE_CORRECTION=TRUE property is found, the duty cycle is adjusted to 50%/50%. If DUTY_CYCLE_CORRECTION=FALSE is found, the duty cycle is unchanged from the original PERIOD specification. If the DUTY_CYCLE_CORRECTION property is not found, the default value of TRUE is assumed.
CLK2X CLK2X180	If originally expressed as FREQUENCY, the FREQUENCY value is doubled. If originally expressed as PERIOD, the PERIOD value is divided in half. The duty cycle is adjusted to 50%/50%.
CLKDV	If originally expressed as FREQUENCY, the FREQUENCY value is divided by the value in the CLKDV_DIVIDE property. If originally expressed as PERIOD, the PERIOD value is multiplied by the value in the CLKDV_DIVIDE property. If the CLKDV_DIVIDE value is not found, the default value of 2.0 is used. The duty cycle is adjusted to 50%.

In addition to creating new PERIOD specifications at the CLKDLL outputs, NGDBuild also creates new TNM or TNM_NET groups to use in those specifications. The new groups are traced forward from the CLKDLL output net to tag all flip-flops, latches, and RAMs controlled by that clock signal.

Each new TNM or TNM_NET group created by NGDBuild is named the same as the corresponding CLKDLL output net.The TS*identifier* for the new PERIOD specification uses the TS_ prefix followed by the net name. These new groups and specifications are shown in timing analysis reports.

The following figure indicates how new groups are created at the outputs of the DLL for a TNM_NET group specification at the CLKDLL inputs.

The sample PERIOD specification for the figure is as follows:

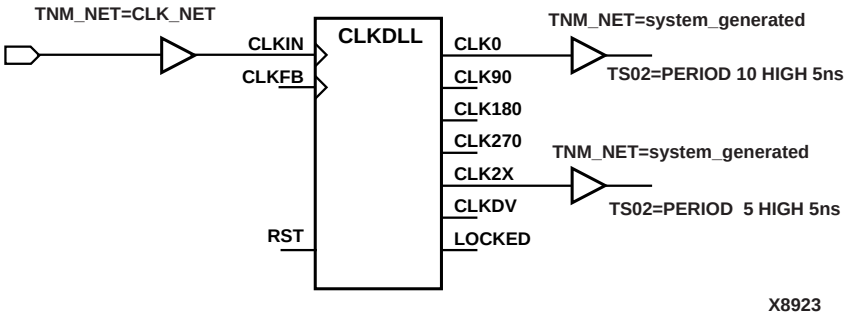


Figure 6-11 TNM_NET Generation

Note The new TNM or TNM_NET groups and PERIOD specifications are not visible in the Constraints Editor, because they are created from the user-applied specification each time NGDBuild is run.

OFFSET Timing Specifications

Offsets are used to define the timing relationship between an external clock and its associated data-in or data-out pin. Using this option allows you to do the following.

- Calculate whether a setup time is being violated at a flip-flop whose data and clock inputs are derived from external nets.
- Specify the delay of an external output net derived from the Q output of an internal flip-flop being clocked from an external device pin.

Following are some of the advantages of using the OFFSET constraint.

- Includes the clock path delay for each individual synchronous elements
- Subtracts the clock path delay from the data path delay for inputs and adds the clock path delay to the data path delay for outputs
- Includes paths for all synchronous element types (FFS, RAMS, and LATCHES)
- Utilizes a global syntax that allows all inputs or outputs to be constrained by a clock
- Allows specifying IO constraints either directly as the setup and clock-to-out required by a device (IN BEFORE and OUT AFTER) or indirectly as the time used by the path external to the device (IN AFTER and OUT BEFORE)

There are basically three types of offset specifications.

- Global
- Net-specific
- Group

Since the global and group OFFSET constraints are not associated with a single data net or component, these two types can also be entered on a TIMESPEC symbol in the design netlist with TS*id*.

Schematic syntax in a TIMESPEC symbol

```
TSid=[TIMEGRP name] OFFSET = {IN|OUT} offset_time
[units] {BEFORE|AFTER} clk_name [TIMEGRP group_name]
```

UCF syntax

```
[TIMEGRP name] OFFSET = {IN|OUT} offset_time [units]
{BEFORE|AFTER} clk_name [TIMEGRP group_name];
```

Note In the UCF file, you cannot specify the TS*id* format.

See the next section and the "Group OFFSET" section for syntax details. As with the PERIOD and MAXDELAY timing specifications, if the same TS*id* is defined in the design netlist (or NCF) and the UCF file, the UCF file takes precedence.

The following subsections describe the use of each type of OFFSET in PCF and UCF files and explain the scope of each specification.

Global OFFSET

Release 3.1i supports the use of the global OFFSET constraint. Release 3.1i also supports the use of time groups within global OFFSET constraints. On a schematic, enter the global OFFSET in the TIMESPEC symbol.

UCF syntax

```
OFFSET = {IN|OUT} offset_time [units] {BEFORE|AFTER}
clk_name [TIMEGRP group_name];
```

PCF syntax

```
OFFSET = {IN|OUT} offset_time [units] {BEFORE|AFTER}
COMP clk_iob_name [TIMEGRP group_name];
```

offset_time is the external offset and *units* is an optional field that indicates the units for the offset time. The default units are nanoseconds, but the timing number can be followed by ps, ns, us, ms, GHz, MHz, or KHz to show the intended units.

The UCF syntax variable *clk_name* is the fully hierarchical net name of the clock net between its pad and its input buffer.

The *clk_iob_name* is the block name of the clock IOB.

The optional TIMEGRP *group_name* defines a group of registers that will be analyzed. By default, all registers clocked by *clk_name* will be analyzed.

IN | **OUT** specifies that the offset is computed with respect to an input IOB or an output IOB. For a bidirectional IOB, the **IN** | **OUT** syntax lets you specify the flow of data (input or output) on the IOB.

BEFORE | **AFTER** indicates whether data is to arrive (input) or leave (output) the device before or after the clock input.

All inputs / outputs are offset relative to *clk_name* or *iob_name*. For example, **OFFSET IN 20 ns BEFORE clk1** dictates that all inputs will

have data present at the pad at least 20 ns before the triggering edge of clk1 arrives at the pad.

Net-Specific OFFSET Constraints

The OFFSET constraint can also be used to specify a constraint for a specific data net in a UCF file or schematic or a specific input or output component in a PCF file.

Schematic syntax when attached to a net

```
OFFSET = {IN|OUT} offset_time [units] {BEFORE|AFTER}
clk_name [TIMEGRP group_name]
```

UCF syntax

```
NET name OFFSET = {IN|OUT} offset_time [units]
{BEFORE|AFTER} clk_name [TIMEGRP group_name];
```

PCF syntax

```
COMP " name" OFFSET = {IN|OUT} offset_time [units]
{BEFORE|AFTER} COMP " clk_iob_name" [TIMEGRP
"group_name"];
```

The PCF file uses blocks (comps) instead of nets.

If **COMP** "iob_name" is omitted in the PCF or NET "name" is omitted in the UCF, the specification is assumed to be global.

offset_time is the external offset.

units is an optional field that indicates the units for offset time. The default units are in nanoseconds, but the timing number can be followed by ps, ns, us, GHz, MHz, or KHz to indicate the intended units.

clk_iob_name is the block name of the clock IOB.

It is possible for one offset constraint to generate multiple data and clock paths (for example, when both data and clock inputs have more than a single sequential element in common).

Examples

The offset constraint examples in this section apply to the following figures.

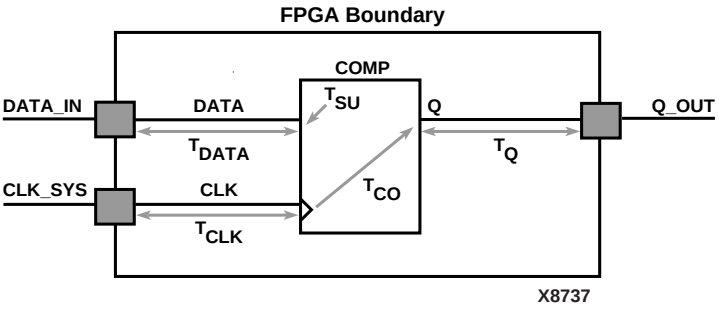


Figure 6-12 OFFSET Example Schematic

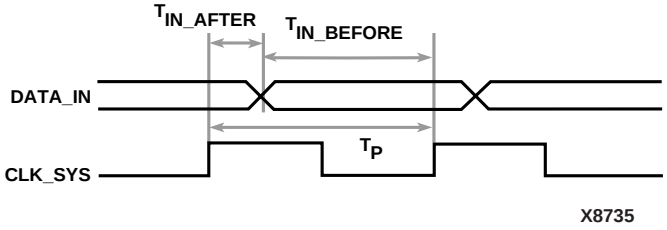


Figure 6-13 OFFSET IN Timing Diagram

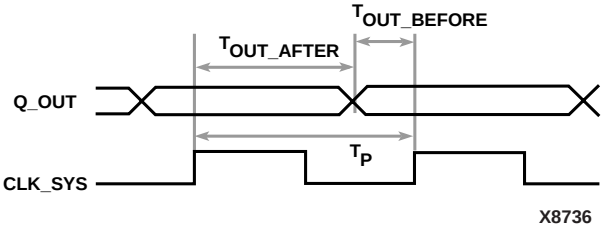


Figure 6-14 OFFSET OUT Timing Diagram

Example 1— OFFSET IN BEFORE

OFFSET IN BEFORE defines the available time for data to propagate from the pad and setup at the synchronous element (COMP). The

time can be thought of as the time differential of data arriving at the edge of the device before the next clock edge arrives at the device. See Figure 6-12 and Figure 6-13. The equation that defines this relationship is as follows.

$T_{DATA} + T_{SU} - T_{CLK} \leq T_{IN\_BEFORE}$

For example, if T_{IN_BEFORE} equals 20 ns, the following syntax applies.

Schematic syntax attached to DATA_IN

```
OFFSET=IN 20.0 BEFORE CLK_SYS
```

UCF syntax

```
NET DATA_IN OFFSET=IN 20.0 BEFORE CLK_SYS;
```

PCF syntax

```
COMP DATA_IN OFFSET=IN 20.0 ns BEFORE COMP CLK_SYS;
```

This constraint indicates that the data will be present on the DATA_IN pad at least 20 ns before the triggering edge of the clock net arrives at the clock pad.

To ensure that the timing requirements are met, the timing analysis software verifies that the maximum delay along the path DATAIN to COMP (minus the 20.0 ns offset) would be less than or equal to the minimum delay along the reference path CLOCK to COMP.

Example 2 — OFFSET IN AFTER

This constraint describes the time used by the data external to the FPGA. OFFSET subtracts this time from the PERIOD declared for the clock to determine the available time for the data to propagate from the pad and setup at the synchronous element. The time can be thought of as the differential of data arriving at the edge of the device after the current clock edge arrives at the edge of the device. See Figure 6-12 and Figure 6-14. The equation that defines this relationship is as follows.

$T_{DATA} + T_{SU} - T_{CLK} \leq T_P - T_{IN\_AFTER}$

T_P is the clock period.

For example, if T_{IN_AFTER} equals 30 ns, the following syntax applies.

Schematic syntax attached to DATA_IN

```
OFFSET=IN 30.0 AFTER CLK_SYS;
```

UCF syntax

```
NET DATA_IN OFFSET=IN 30.0 AFTER CLK_SYS;
```

PCF syntax

```
COMP DATA_IN OFFSET=IN 30.0 ns AFTER COMP CLK_SYS;
```

This constraint indicates that the data will arrive at the pad of the device (COMP) no more than 30 ns after the triggering edge of the clock arrives at the clock pad. The path DATA_IN to COMP would contain the setup time for the COMP data input relative to the CLK_SYS input.

Verification is almost identical to Example 1, except that the offset margin (30.0 ns) is added to the data path delay. This is caused by the data arriving after the reference input. The timing analysis software verifies that the data can be clocked in prior to the next triggering edge of the clock.

A PERIOD or FREQUENCY is required only for offset **OUT** constraints with the **BEFORE** keyword or offset **IN** with the **AFTER** keyword.

Example 3 — OFFSET OUT AFTER

This constraint defines the time available for the data to propagate from the synchronous element to the pad. This time can also be considered as the differential of data leaving the edge of the device after the current clock edge arrives at the edge of the device. See Figure 6-12 and Figure 6-14.

The equation that defines this relationship is as follows.

$T_Q + T_{CO} + T_{CLK} \leq T_{OUT\_AFTER}$

For example, if T_{OUT_AFTER} equals 35 ns, the following syntax applies.

Schematic syntax attached to Q_OUT

```
OFFSET=OUT 35.0 AFTER CLK_SYS
```

UCF syntax

```
NET Q_OUT OFFSET=OUT 35.0 AFTER CLOCK;
```

PCF syntax

```
COMP Q_OUT OFFSET=OUT 35.0 ns AFTER COMP CLK_SYS;
```

This constraint calls for the data to leave the FPGA 35 ns after the present clock input arrives at the clock pad. The path COMP to Q_OUT would include the CLOCK-to-Q delay (component delay).

Verification involves ensuring that the maximum delay along the reference path (CLK_SYS to COMP) and the maximum delay along the data path (COMP to Q_OUT) does not exceed the specified offset.

Example 4 — OFFSET OUT BEFORE

This constraint defines the time used by the data external to the FPGA. OFFSET subtracts this time from the clock PERIOD to determine the available time for the data to propagate from the synchronous element to the pad. The time can also be considered as the differential of data leaving the edge of the device before the next clock edge arrives at the edge of the device. See Figure 6-12 and Figure 6-14. The equation that defines this relationship is as follows.

$T_Q + T_{CO} + T_{CLK} \leq T_P - T_{OUT\_BEFORE}$

For example, if T_{OUT_BEFORE} equals 15 ns, the following syntax applies.

Schematic syntax attached to Q_OUT

```
OFFSET=OUT 15.0 BEFORE CLK_SYS
```

UCF syntax

```
NET Q_OUT OFFSET=OUT 15.0 BEFORE CLK_SYS;
```

PCF syntax

```
COMP Q_OUT OFFSET=OUT 15.0 ns BEFORE COMP CLK_SYS;
```

This constraint states that the data clocked to Q_OUT must leave the FPGA 15 ns before the next triggering edge of the clock arrives at the clock pad. The path COMP to Q_OUT includes the CLK_SYS-to-Q delay (component delay). The data clocked to Q_OUT will leave the FPGA 15.0 ns before the next clock input.

Verification involves ensuring that the maximum delay along the reference path (CLK_SYS to COMP) and the maximum delay along the data path (COMP to Q_OUT) do not exceed the clock period minus the specified offset.

As in Example 2, a PERIOD or FREQUENCY constraint is required only for offset OUT constraints with the BEFORE keyword or offset IN with the AFTER keyword.

Specific OFFSET Constraints with Timegroups

A clock register time group allows you to define a specific set of registers to which an OFFSET constraint applies based on a clock edge. Consider the following example.

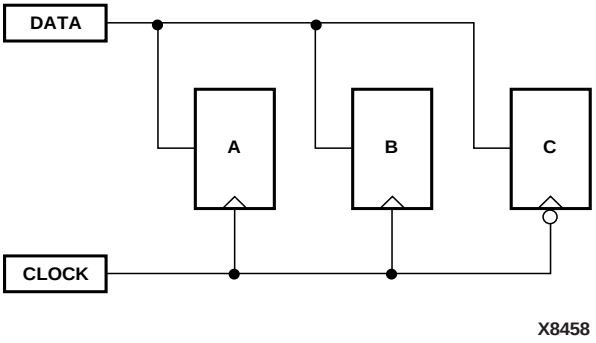


Figure 6-15 Using Timegroups with Registers

You can define time groups for the registers A, B and C, even though these registers have the same data and clock source. The syntax is as follows.

Schematic syntax in TIMEGRP primitive

```
AB=RISING FFS C =FALLING FFS;
```

UCF / PCF syntax

```
TIMEGRP AB=RISING FFS; TIMEGRP C =FALLING FFS;
```

Schematic syntax attached to DATA

```
OFFSET=IN 10 BEFORE CLOCK TIMEGRP AB
OFFSET=IN 20 BEFORE CLOCK TIMEGRP C
```

UCF syntax

```
NET DATA OFFSET=IN 10 BEFORE CLOCK TIMEGRP AB;
NET DATA OFFSET=IN 20 BEFORE CLOCK TIMEGRP C;
```

PCF syntax

```
COMP DATA OFFSET=IN 10 BEFORE COMP CLOCK TIMEGRP AB;
COMP DATA OFFSET=IN 20 BEFORE COMP CLOCK TIMEGRP C;
```

Even though the registers A, B and C have a common data and clock source, timing analysis applies two different offsets (10 ns and 20 ns). Registers A and B belong to the offset with 10 ns and Register C belongs to the offset with 20 ns.

However, you must use some caution when using timegroups with registers. Consider the following diagram.

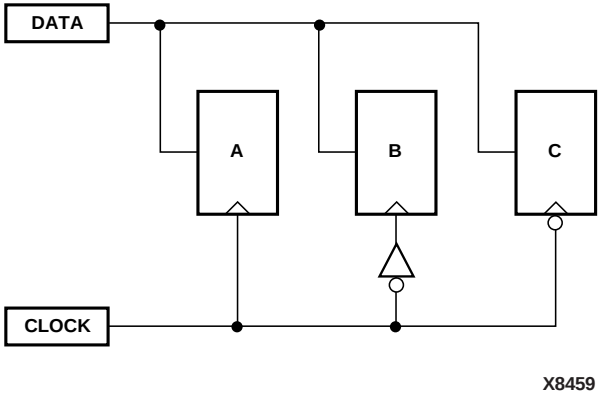


Figure 6-16 Problematic Timegroup Definition

This circuit is identical to Figure 6-15 except that an inverter has been inserted in the path to Register B. In this instance, even though this register is a member of the time group whose offset triggers on the

rising edge, the addition of the inverter classifies register B as triggering on the falling edge like Register C.

Normally, the tools will move an inverter to the register, in which case, B would be a part of the timegroup "Falling". However if the clock is gated with logic that inverts, then the inverter will not become part of the register. In that case, one way to solve this problem is to create a timegroup with an exception for Register B. See the "Creating Groups by Exclusion" section for details.

Group OFFSET

You can also define OFFSET constraints within the TIMESPEC primitive with a leading TIMEGRP reference.

Schematic syntax in TIMESPEC primitive

TS*identifier***=TIMEGRP** *name* **OFFSET=**{**IN**|**OUT**} *offset_time* [*units*] {**BEFORE**|**AFTER**} *clk_name* [**TIMEGRP** *group_name*]

The UCF and PCF syntax do not require the **TS***identifier*.

UCF syntax

[**TIMEGRP** *name***] OFFSET=** {**IN**|**OUT**} *offset_time* [*units*] {**BEFORE**|**AFTER**} *clk_name* [**TIMEGRP** *group_name*]**;**

PCF syntax

[**TIMEGRP** *name***] OFFSET=** {**IN**|**OUT**} *offset_time* [*units*] {**BEFORE**|**AFTER**} **COMP** *clk_iob_name* [**TIMEGRP** *group_name*]**;**

The timing group specified at the beginning has a different purpose than the timegroup specified at the end. The first time group is a list of data pads that the OFFSET applies to, while the last time group (register time group) is a list of synchronous elements that specifies which register elements clocked by *clk_name* or *clk_iob_name* should be analyzed.

Note If the first group has FFs or the second group has PADS, NGDBuild generates an error.

offset_time is the external offset.

units is an optional field that indicates the units for offset time. The default units are in nanoseconds, but the timing number can be

followed by ps, ns, us, GHz, MHz, or KHz to indicate the intended units.

clk_iob_name is the block name of the clock IOB.

Ignoring Selected Paths (TIG)

In a design, some paths do not require timing analysis. These are paths that exist in the design, but are never used during time-critical operations. If you indicate a timing requirement on these paths, more important paths might be slower, which can result in failure to meet the timing requirements.

The value of TIG may be any of the following.

- Empty (global TIG that blocks all paths)
- A single TSid to block
- A comma separated list of TSids to block, for example

```
NET $1I567/$Sig_5 TIG=TS_fast, TS_even_faster;
```

To indicate that all timing specifications through a net, primitive pin or macro pin are to be ignored, attach the following attribute to the desired element.

Schematic syntax

TIG

UCF syntax

{**NET** | **PIN** | **INSTANCE**} *name* **TIG;**

If this attribute is attached to a net, primitive pin, or macro pin, all paths that fan forward from the point of application of the attribute are treated as if they don't exist for the purposes of timing analysis during implementation. In the following figure, NET C is ignored. However, note that the lower path of NET B that runs through the two OR gates would not be ignored.

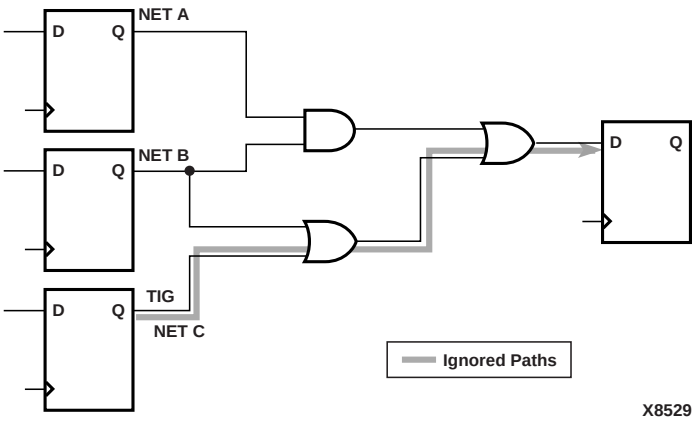


Figure 6-17 TIG Example

The following attribute would be attached to a net to inform the timing analysis tools that it should ignore paths through the net for specification TS43:

Schematic syntax

```
TIG = TS43
```

UCF syntax

```
NET net_name TIG = TS43;
```

You cannot perform path analysis in the presence of combinatorial loops. Therefore, the timing tools ignore certain connections to break combinatorial loops. You can use the TIG constraint to direct the timing tools to ignore specified nets or load pins, consequently controlling how loops are broken.

Basic FROM –TO Syntax

Within the TIMESPEC primitive, you use the following syntax to specify timing requirements between specific end points.

```
TSidentifier=FROM source_group TO dest_group delay
TSidentifier=FROM source_group delay
TSidentifier =TO dest_group delay
```

Unspecified FROM or TO, as in the second and third syntax statements, implies all points.

The From-To statements are TS attributes that reside in the TIMESPEC primitive. The parameters *source_group* and *dest_group* must be one of the following.

- Predefined groups
- Previously created TNM identifiers
- Groups defined in TIMEGRP symbols
- TPSYNC groups

Predefined groups consist of FFS, LATCHES, RAMS, or PADS and are discussed in the "Using Predefined Groups" section. TNMs are introduced in the "Creating User-Defined Groups Using TNMs" section. TIMEGRP symbols are introduced in the "Creating New Groups from Existing Groups" section.

Note Keywords, such as FROM, TO, and TS appear in the documentation in upper case; however, you can enter them in the TIMESPEC primitive in either upper or lower case. You cannot enter them in a combination of lower and upper case.

The *delay* parameter defines the maximum delay for the attribute. Nanoseconds are the default units for specifying delay time in TS attributes. You can also specify delay using other units, such as picoseconds or megahertz.

Refer to the "Specifying Time Delay in TS Attributes" section later in this chapter for more information on time delay. The delay can be a function of another TIMESPEC (TS01*2).

The following examples illustrate the use of From-To TS attributes.

Schematic syntax in TIMESPEC primitive

```
TS01=FROM FFS TO FFS 30
TS_OTHER=FROM PADS TO FFS 25
TS_THIS=FROM FFS TO RAMS 35
TS_THAT=FROM PADS TO LATCHES 35
```

UCF syntax

```
TIMESPEC TS01=FROM FFS TO FFS 30;
TIMESPEC TS_OTHER=FROM PADS TO FFS 25;
TIMESPEC TS_THIS=FROM FFS TO RAMS 35;
```

```
TIMESPEC TS_THAT=FROM PADS TO LATCHES 35;
```

You can place TS attributes containing From-To statements in either of two places: in the TIMESPEC primitive on the schematic as discussed in this chapter or in a constraints (UCF) file.

Specifying Timing Points

There are situations where a particular point or set of points in your design needs to be flagged for reference in subsequent timing specifications. *Timing points* are used for these specifications.

There are two types of timing points.

- A TPSYNC timing point is used to allow a point to be used as the start or the end of timing path, even though the point may not apply to a flip-flop, latch, RAM or I/O pad.
- A TPTHRU timing point identifies an intermediate point on a path.

The following sections describe how these timing points are specified.

Using TPSYNC to Define Synchronous Points

There are cases where the timing of a design must be defined from or to a point in the design that is not a flip-flop, latch, RAM or I/O pad. For example, you might want to specify a point at the output of a latch defined using a function generator instead of a latch symbol. The TPSYNC timing point identifies one or a group of these points.

A TPSYNC attribute has the following syntax.

Schematic syntax

TPSYNC = *identifier*

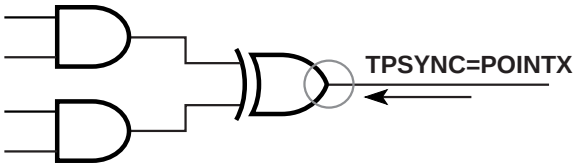
UCF syntax

{**NET** | **PIN** | **INST**} **TPSYNC=** *identifier*;

identifier is a name that is used in timing specifications in the same way that groups are used. The same identifier can be used on several points which are then treated as a group from the point of view of timing analysis. The *identifier* must be different from any identifier used for a TNM attribute.

The way a TPSYNC timing point is used depends on the object to which it is attached.

- Attached to a net, TPSYNC identifies the source of the net as a potential source or destination for timing specifications.



X8524

- Attached to an output macro pin, TPSYNC identifies all of the sources inside the macro that drive the pin to which the attribute is attached as potential sources or destinations for timing specifications. In the following diagram. POINTY applies to the inverter.

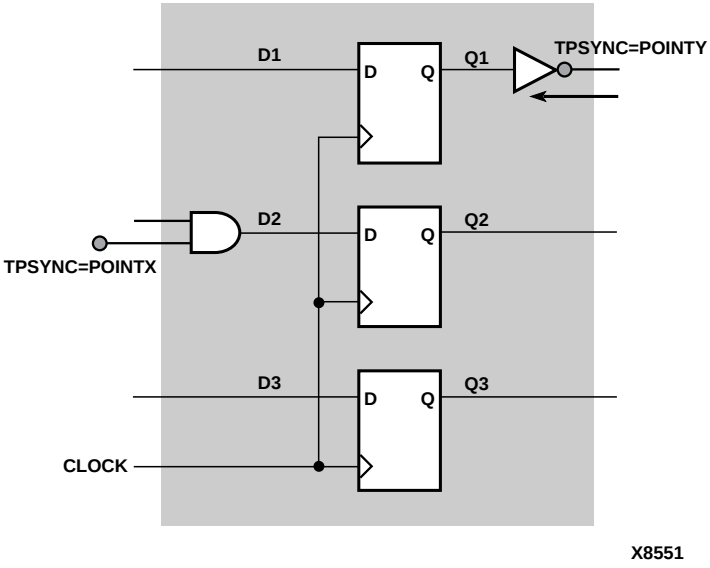
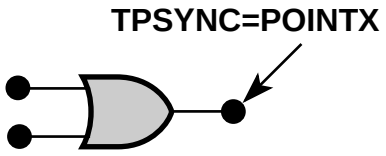


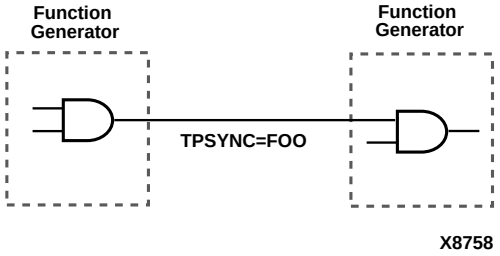
Figure 6-18 TPSYNCs Attached to Macro Pins

If the macro pin is an input pin (that is, there are no sources for the pin in the macro), then all of the load pins in the macro are flagged as synchronous points. In the preceding figure POINTX applies to the input AND gate.

- Attached to a primitive pin, TPSYNC flags the primitive's pin as a potential source or destination for timing specifications; TPSYNC applies to the pin it is attached to.
- Attached to a primitive symbol, TPSYNC identifies the output(s) of that element as a potential source or destination for timing specifications. See the following figure.



The use of a TPSYNC timing point to define a synchronous point in a design implies that the flagged point cannot be merged into a function generator. For example, consider the following diagram.



In this example, because of the TPSYNC definition, the two gates cannot be merged into a single function generator.

Using TPTHRU to Define Through Points

The TPTHRU attribute defines an intermediate point in a path. A point or group defined with TPTHRU attributes is used in detailed timing specifications.

A TPTHRU attribute has the following syntax.

TPTHRU = *identifier*

identifier is a name that is used in timing specifications in the same way that groups are used. The same identifier can be used on several points which are then treated as a group from the point of view of timing analysis.

The *identifier* must be different from any identifier used for a TNM attribute or TPSYNC.

Timing specifications using TPTHRU groups are described in the "Specifying Time Delay in TS Attributes" section.

Using TPTHRU or TPSYNC in a FROM–TO Constraint

It is sometimes convenient to define intermediate points on a path to which a specification applies. This defines the maximum allowable delay and has the following syntax.

Schematic syntax in TIMESPEC primitive

TS*identifier***=FROM** *source_group* **THRU** *thru_point* [**THRU** *thru_point*] **TO** *dest_group allowable_delay* [*units*]

TS*identifier***=FROM** *source_group* **THRU** *thru_point* [**THRU** *thru_point*] *allowable_delay* [*units*]

TS*identifier***=THRU** *thru_point* **[THRU** *thru_point***] TO** *dest_group allowable_delay* [*units*]

UCF syntax

TIMESPECTS*identifier***=FROM** *source_group* **THRU** *thru_point* [**THRU** *thru_point*] **TO** *dest_group allowable_delay* [*units*];

TIMESPEC TS*identifier***=FROM** *source_group* **THRU** *thru_point* [**THRU** *thru_point*] *allowable_delay* [*units*];

TIMESPEC TS*identifier***=THRU** *thru_point* **[THRU** *thru_point***]** allowable_delay [*units*];

Unspecified FROM or TO, as in the second and third syntax statements, implies all points.

- *identifier* is an ASCII string made up of the characters A..Z, a..z, 0..9, underbar (_), and forward slash (/).
- *source_group* and *dest_group* are user-defined, predefined groups or TPSYNCs.
- *thru_point* is an intermediate point used to qualify the path, defined using a TPTHRU attribute.
- *allowable_delay* is the timing requirement.
- *units* is an optional field to indicate the units for the allowable delay. Default units are nanoseconds, but the timing number can

be followed by ps, ns, us, ms, GHz, MHz, or KHz to indicate the intended units.

The example shows how to use the TPTHRU attribute with the THRU attribute on a schematic. The UCF syntax is as follows.

```
INST FLOPA TNM=A;

INST FLOPB TNM=B;
NET MYNET TPTHRU=ABC
TIMESPEC TSpath1=FROM A THRU ABC TO B 30;
```

The following schematic shows the placement of the TPTHRU attribute and the resultant path that is defined.

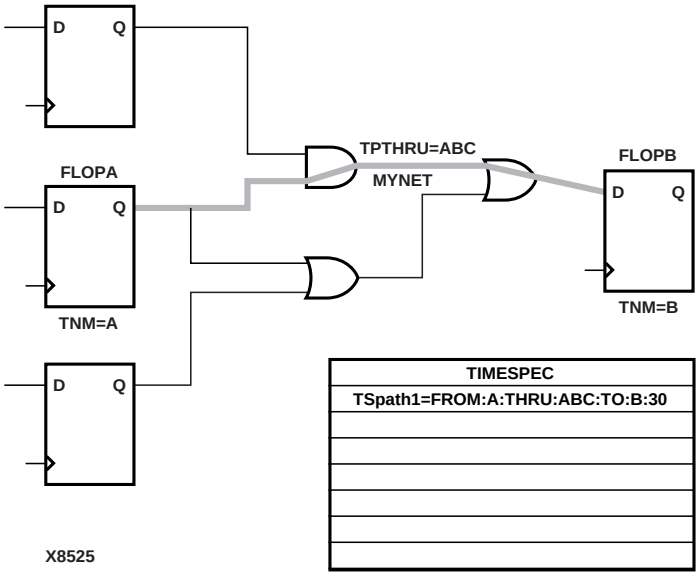


Figure 6-19 TPTHRU Example

Specifying Time Delay in TS Attributes

Nanoseconds are the default units for specifying delay times in TS attributes. However, after specifying the maximum delay or minimum frequency numerically, you can enter the unit of measure by specifying the following.

- PS for picoseconds, NS for nanoseconds, US for microseconds, or MS for milliseconds
- MHZ for megahertz, KHZ for kilohertz, or GHz for gigahertz

As an alternate way of specifying time delay, you can specify one time delay in terms of another. Instead of specifying a time or frequency in a TS attribute definition, you can specify a multiple or division of another TS attribute. This is useful in a system where all clocks are derived from a master clock; in this situation, changing the timing specification for the master clock changes the specification for all clocks in the system.

Use the syntax below to specify a TS attribute delay in terms of another.

Schematic syntax attached to TIMESPEC primitive

TS*identifier***=***specification*
reference_TS_attribute[{*****|**/**}*number*]

UCF syntax

TIMESPEC TS*identifier***=***specification*
reference_TS_attribute[{*****|**/**}*number*];

number can be either a whole number or a decimal. The *specification* can be any From-To statement as illustrated by the following examples.

```
FROM PADS TO PADS

FROM group1 TO group2

FROM tnm_identifier TO FFS

FROM LATCHES TO group1
```

Use "*" to represent multiplication and "/" to represent division. The specification type of the reference TS attribute does not need to be the same as the TS attribute being defined; however, it must not be specified in terms of TIG.

Examples:

Examples of specifying a TS attribute in terms of another are as follows. In these cases, assume that the reference attributes were specified as delays (not frequencies).

In the example below, the paths between flip-flops and pads are placed and routed so that their delay is at most 10 times the delay specified in the TS05 attribute.

Schematic syntax in TIMESPEC primitive

```
TS08=FROM FFS TO PADS TS05*10
```

UCF syntax

```
TIMESPEC TS08=FROM FFS TO PADS TS05*10;
```

In the example below, the paths between input and output pads are placed and routed so that their delay is at most one-eighth the delay specified in the TS07 attribute.

Schematic syntax in TIMESPEC primitive

```
TS1=FROM PADS TO PADS TS07/8
```

UCF syntax

```
TIMESPEC TS1=FROM PADS TO PADS TS07/8;
```

Note When a reference attribute is specified as a frequency, a multiple represents a faster specification; a division represents a slower specification.

You can also specify a TS attribute in terms of a TS attribute that is already a specification of another. The following example provides an illustration.

Schematic syntax in TIMESPEC primitive

```
TS09=FROM FFS TO FFS 50
TS10=FROM FFS TO PADS TS09*2
TS11=FROM PADS TO PADS TS10*4
```

UCF syntax

```
TIMESPEC TS09=FROM FFS TO FFS 50;
TIMESPEC TS10=FROM FFS TO PADS TS09*2;
TIMESPEC TS11=FROM PADS TO PADS TS10*4;
```

Using the PRIORITY Keyword

There may be situations where there is a conflict between two TIMESPECs that cover the same path. In these cases you can define the priority of a TIMESPEC using the following syntax.

```
normal_timespec_syntax PRIORITY integer
```

normal_timespec_syntax is a legal TIMESPEC and *integer* represents the priority (the smaller the number, the higher the priority). The number can be positive, negative, or zero, and the value only has meaning when compared with other PRIORITY values.

Note The PRIORITY keyword cannot be used with the MAXDELAY, or MAXSKEW constraint.

Sample Schematic Using TIMESPEC/TIMEGRP Symbol

TNM identifiers define symbols or groups of symbols that are used in timing specifications. They can also define other groups. The following figure shows an example of a TNM attribute attached to an individual symbol. In this circuit, the flip-flop D_FF has the attribute TNM=D_FF attached to it.

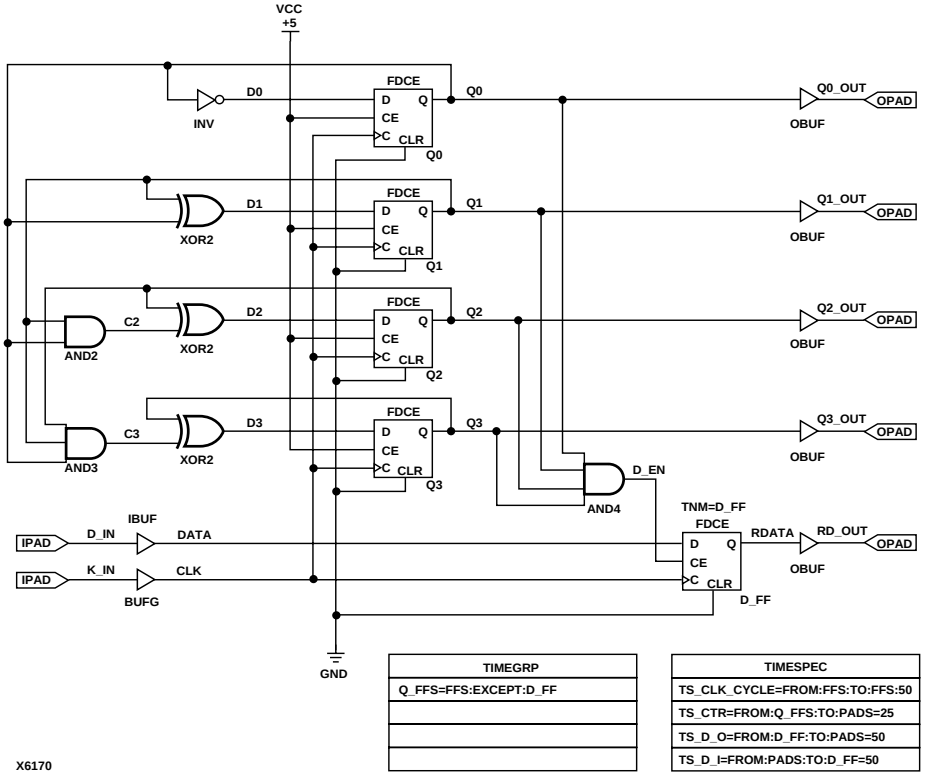


Figure 6-20 Example of Using TNMs and TIMEGRPs in a Schematic

The TIMEGRP symbol contains an attribute that defines a group of flip-flops called Q_FFS, which includes all flip-flops in the schematic except the one labeled D_FF. You can then use the group Q_FFS to create timing specifications in the TIMESPEC primitive. The flip-flop D_FF has its clock enable driven at 1/2 of the clock frequency; therefore, its flip-flop to pad and pad to flip-flop timing specifications are longer than the flip-flop to pad specifications in the Q_FFS group.

Prorating Constraints

The prorating constraints, VOLTAGE and TEMPERATURE, provide a method for determining timing delay characteristics based on known environmental parameters. On a schematic, you can enter these constraints in any empty space. For Release 3.1i these two constraints are supported for XC4000XL / XLA. New speed file releases for existing architectures will support these two constraints.

Note Prorating is not intended for military and industrial ranges. It is applicable only within the commercial ranges.

VOLTAGE Constraint

This constraint allows the specification of the operating voltage. This provides a means of prorating delay characteristics based on the specified voltage.

Note Each architecture has its own specific range of supported voltages. If the entered voltage does not fall within the supported range, the constraint is ignored and an architecture-specific default value is used instead. The UCF syntax is as follows.

VOLTAGE=*value*[*units*]

value is an integer or real number specifying the voltage and units is an optional parameter specifying the unit of measure. V specifies volts, the default voltage unit.

TEMPERATURE Constraint

This constraint allows the specification of the operating temperature which provides a means of prorating device delay characteristics based on the specified junction temperature. Prorating is a linear scaling operation on existing speed file delays and is applied globally to all delays.

Note Each architecture has its own specific range of valid operating temperatures. If the entered temperature does not fall within the supported range, the constraint is ignored and an architecture-specific default value is used instead. The UCF syntax is as follows.

TEMPERATURE=*value*[**C |F| K**]

value is an integer or a real number specifying the temperature. C, K, and F are the temperature units: F is degrees Fahrenheit, K is degrees Kelvin, and C is degrees Celsius, the default.

Additional Timing Constraints

There are additional properties and constraints you can specify for the timing analysis tools. They are the following.

- Net skew control (MAXSKEW)
- Net delay control
- Path tracing control
- The DROP_SPEC constraint
- The USELOWSKEWLINES constraint

Controlling Net Skew (MAXSKEW)

Skew is the difference between the minimum and maximum of the maximum load delays on a net. You can control the maximum allowable skew on a net by attaching the MAXSKEW attribute directly to the net. Syntax is as follows.

```
skew_item MAXSKEW=allowable_skew [units];
```

allowable_skew is the timing requirement.

The default units for *allowable_skew* are nanoseconds, but the timing number can be followed by ps, ns, us, ms, GHz, MHz, or KHz to indicate the intended units.*skew_item* is one of the following,

- NET “*net_name*”
- TIMEGRP “*group_name*” (PCF only)
- ALLCLOCKNETS (PCF only)

Note TIMEGRP and ALLCLOCKNETS are supported in PCF files only.

It is important to understand exactly what MAXSKEW defines. Consider the following example.

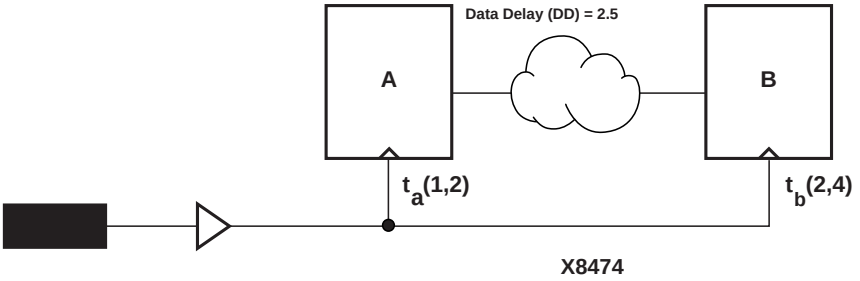


Figure 6-21 MAXSKEW

In the preceding diagram, for t_a(1,2), 1 ns is the minimum delay and 2 ns is the maximum delay for the Register A clock. For t_b(2,4), 2 ns is the minimum delay and 4 ns is the maximum delay for the Register B clock. MAXSKEW defines the maximum of t_b minus the maximum of t_a, that is, 4-2=2. Since the data delay is greater than MAXSKEW (DD is 2.5 while MAXSKEW is 2), no race condition occurs. However, MAXSKEW does not account for the circumstance where one of the registers is operating at minimum delay (for example, t_a=1) while a second register is operating at maximum delay (for example, t_b=4). Under those conditions, the skew is 3 ns (t_b - t_a= 3). Since the data delay (DD = 2.5) is less than the skew, a race condition exists.

Controlling Net Delay (MAXDELAY)

You can control the maximum allowable delay on a net by attaching the MAXDELAY attribute directly to the net. The UCF syntax is as follows.

NET *net_name* **MAXDELAY=***path_value*

path_value is one of the following,

- *delay_time [units]*

  units defaults to nanoseconds, but the delay time number can be followed by ps, ns, us, or ms (picoseconds, nanoseconds, micro-seconds, or milliseconds) to specify the units

- *frequency units*

  units can be specified as GHz, MHz, or KHz (gigahertz, mega-hertz, or kilohertz*)*

Controlling Path Tracing

Path tracing controls allows you to enable or disable specific paths within device components (for example, CLBs and IOBs) for timing analysis. *These constraints can only be entered in a PCF file;* they cannot be applied during design entry or in a UCF or NCF file.

This constraint can be applied at a global or group scope. The path tracing syntax is as follows.

[**TIMEGRP** *predefined_group*] {**ENABLE** | **DISABLE**}**=** *symbol*;

symbol is a component delay symbol, and *predefined_group* (which is optional) represents the name of a previously-defined time group. If there is no TIMEGRP *predefined_group* qualifier, the path tracing control applies to all logic cells in the design.

The *symbol*, which is case-insensitive, can be either of the following.

- A standard component delay symbol name (for example, reg_sr_q or tbuf_i_o, as described in the following table).
- There is a one-to-many correspondence between these symbol names and data book symbol names, and the data book symbols to which each standard block delay signal applies varies from one device family to another.
- A component delay specified in the *Xilinx Programmable Logic Data Book* (for example, T_{ILO} (entered as TILO) or T_{CCK} (entered as TCCK)).

The following table describes the standard block delay symbols.

Table 6-1 Standard Block Delay Symbols for Path Tracing

Symbol	Path Type	Default
reg_sr_q	Set/Reset to output propagation delay	Disabled
reg_sr_clk	Set/Reset to clock setup and hold checks	Disabled
lat_d_q	Data to output transparent latch delay	Disabled
ram_d_o	RAM data to output propagation delay	Disabled

Table 6-1 Standard Block Delay Symbols for Path Tracing

Symbol	Path Type	Default
ram_we_o	RAM write enable to output propagation delay	Enabled
tbuf_t_o	TBUF tristate to output propagation delay	Enabled
tbuf_i_o	TBUF input to output propagation delay	Enabled
io_pad_i	IO pad to input propagation delay	Enabled
io_t_pad	IO tristate to pad propagation delay	Enabled
io_o_i	IO output to input propagation delay. Disabled for tristated IOBs	Enabled
io_o_pad	IO output to pad propagation delay	Enabled

The IOB configuration for Virtex/-E/-II and Spartan-II is somewhat different than the IOB configuration for other architectures. See the following figure

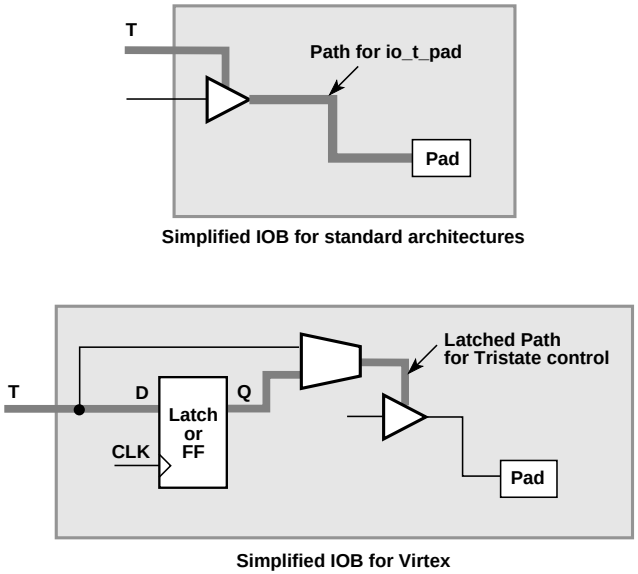


Figure 6-22 Simplified IOB Configurations and io_t_pad

For the Virtex / -E / -II and Spartan-II IOBs, there is no default path. If a latch is used (latch mode), then io_t_pad controls the D to Q path through the latch. By default D to Q is enabled which is different than other internal latches. The clock to Q of the latch is not disabled by io_t_pad.

If a register is used instead of a latch, the clock to Q of the register is not disabled by io_t_pad.

Path Tracing Examples

The PCF file constraint below prevents timing analysis on any path that includes the I to O delay on a TBUF component. The constraint applies to all TBUF components in the design.

```
DISABLE = "tbuf_i_o";
```

The PCF file constraint below disables the I to O delay on the TBUF components in the group mygroup, if applicable.

```
TIMEGRP "mygroup" DISABLE = "tbuf_i_o";
```

The PCF file constraint below disables the T_{ILO} databook component delay in the group mygroup, if applicable.

```
TIMEGRP "mygroup" DISABLE = "TILO";
```

The delay symbol names in the *Xilinx Programmable Logic Data Book* do not always agree with the delay names reported in TRACE (the Xilinx timing analyzer). To ensure your path tracing constraints are processed correctly and to allow your constraints to be portable from one device to another, use the delay names reported by TRACE instead of the databook names.

You can control path tracing for a single instance by creating a group containing only the instance, then specifying this group in a path tracing constraint.

The DROP_SPEC Constraint

A constraint specified in a UCF constraints file takes precedence over one with the same name in the input design. This allows you to redefine or modify constraints without having to edit the input design. The DROP_SPEC constraint allows you to specify that a timing constraint defined in the input design should be dropped from the analysis. The UCF syntax is as follows.

TS*identifier* **= DROP_SPEC**

identifier is the identifier name used with another timing specification. This constraint can be used when new specifications defined in a constraints file do not directly override all specifications defined in the input design, and some of these input design specifications need to be dropped.

While this timing command is not expected to be used much in an input netlist (or NCF file), it is not illegal. If defined in an input design this attribute must be attached to a TIMESPEC primitive.

The USELOWSKEWLINES Constraint

The Virtex/-E/-II and Spartan-II devices offer twenty four lines of low skew resources for secondary clocks. You can specify the USELOWSKEWLINES constraint in the UCF file to take advantage of these resources. Specify this constraint only when all four primary clocks have been used.

The UCF syntax for the USELOWSKEWLINES constraint follows.

```
NET net_name USELOWSKEWLINES;
```

The presence of a USELOWSKEWLINES constraint on any net indicates that this net must be routed on secondary low-skew clock resources. PAR routes the low skew resources that trigger the timing analyzer to perform a skew analysis.

Constraints Priority

In some cases, two timing specifications cover the same path. For cases where the two timing specifications on the path are mutually exclusive, the following constraint rules apply.

- Priority depends on the file in which the constraint appears. A constraint in a file accessed later in the design flow replaces a constraint in a file accessed earlier in the design flow. Priority is as follows (first listed is the highest priority, last listed is the lowest).
  - Constraints in a Physical Constraints File (PCF)
  - Constraints in a User Constraints File (UCF)
  - Constraints in a Netlist Constraints File (NCF)
  - Attributes in a schematic
- If two timing specifications cover the same path, the priority is as follows (first listed is the highest priority, last listed is the lowest).
  - Timing Ignore (TIG)
  - FROM THRU TO
  - FROM TO
  - Specific OFFSET
  - Group OFFSET
  - Global OFFSET
  - PERIOD
  - ALLPATHS
- FROM THRU TO or FROM TO statements have a priority order that depends on the type of source and destination groups included in a statement. The priority is as follows (first listed is the highest priority, last listed is the lowest).
  - Both the source group and the destination group are user-defined groups
  - Either the source group or the destination group is a predefined group

- Both the source group and the destination group are predefined groups

- OFFSET constraints take precedence over more global constraints such as the ALLPATHS constraints.

  If two specific OFFSET constraints at the same level of precedence interact, an OFFSET with a register qualifier takes precedence over an OFFSET without a qualifier; if otherwise equivalent, the latter in the constraint file takes precedence.

- Net delay and Net skew specifications are analyzed independently of path delay analysis and do not interfere with one another.

Syntax Summary

The following sections summarize the syntax for timing constraints.

TNM Attributes

The following table lists the syntax used when creating TNMs, which you enter directly on the primitive symbol, macro symbol, net, or driver pin.

TNM Attribute Syntax	Where Applied
Schematic syntax: **TNM=***identifier* **TNM=***predefined_group identifier* UCF syntax: {**NET** | **PIN** | **INSTANCE**} *name* **TNM=***identifier* {**NET** | **PIN** | **INSTANCE**} *name* **TNM=***predefined_group identifier***;**	Net, Symbol, Pin, Macro

TIMEGRP Attributes

The following table lists the syntax used with the TIMEGRP primitive.

Group Type	TIMEGRP Attribute Syntax
Combine	Schematic syntax in TIMEGRP primitive: *new_group*=*group1 group2* [*group3* . . .] UCF syntax: TIMEGRP *new_group*=*group1 group2* [*group3* . . .];
Exclude	Schematic syntax in TIMEGRP primitive: *new_group*=*group1*[*group2* . . .]**EXCEPT** *group3*[*group4* ...] UCF syntax: **TIMEGRP** *new_group*=*group1*[*group2* . . .] **EXCEPT** *group3*[*group4* ...];
Clock Edge (flip-flops)	Schematic syntax in TIMEGRP primitive: *new_group*=**RISING** *group1* *new_group*=**FALLING** *group1* UCF syntax: **TIMEGRP** *new_group*=**RISING** *group1*; **TIMEGRP** *new_group*=**FALLING** *group1*;
Gate Edge (latches)	Schematic syntax in TIMEGRP primitive: *new_group*=**TRANSHI** *group1* *new_group*=**TRANSLO** *group1* UCF syntax: TIMEGRP *new_group*=**TRANSHI** *group1*; TIMEGRP *new_group*=**TRANSLO** *group1*;

Group Type	TIMEGRP Attribute Syntax
Pattern Matching	Schematic syntax in TIMEGRP primitive: *new_group=predefined_group* **(***name_qualifier1*[*name_qualifier2 . . .*]**)** UCF syntax: **TIMEGRP** *new_group=predefined_group* **(***name_qualifier1*[*name_qualifier2 . . .*]**);**
Net-specific OFFSETs	Schematic syntax when attached to a net: **OFFSET =** {**IN** | **OUT**} *offset_time* [*units*] {**BEFORE** | **AFTER**} *clk_name* [**TIMEGRP** *group_name*] UCF syntax: **NET** *name* **OFFSET =** {**IN** | **OUT**} *offset_time* [*units*] {**BEFORE** | **AFTER**} *clk_name* [**TIMEGRP** *group_name*]**;** PCF syntax: **COMP "***iob_name***" OFFSET =** {**IN** | **OUT**} *offset_time* [*units*] {**BEFORE** | **AFTER**} **COMP "***clk_iob_name***"** [**TIMEGRP "**group_name**"];**

TIMESPEC Attributes

The following table lists the syntax used for parameters that define TS attributes, which reside in the TIMESPEC primitive or appear in UCF or NCF files.

Spec Type	TS Attribute Syntax
Basic From-To	Schematic syntax in TIMESPEC primitive: **TS***id***=FROM** *source_group* **TO** *dest_group delay* **TS***id***=FROM** *source_group delay* **TS***id***=TO** *dest_group delay* UCF syntax: **TIMESPEC TS***id***=FROM** *source_group* **TO** *dest_group delay***;** **TIMESPEC TS***id***=FROM** *source_group delay***;** **TIMESPEC TS***id***=TO** *dest_group delay***;**
Ignore	Schematic syntax in TIMESPEC primitive: **TS***id***=IGNORE** UCF syntax: **TIMESPEC TS***id***=IGNORE;**
Through point	Schematic syntax in TIMESPEC primitive: **TS***id***=FROM** *source_group* **THRU** *thru_point*[**THRU** *thru_point*] **TO** *dest_group delay* **TS***id***=FROM** *source_group* **THRU** *thru_point*[**THRU** *thru_point*] *delay* **TS***id***=THRU** *thru_point*[**THRU** *thru_point*] **TO** *dest_group delay* UCF syntax: **TIMESPEC TS***id***=FROM** *source_group* **THRU** *thru_point*[**THRU** *thru_point*] **TO** *dest_group delay***;** **TIMESPEC TS***id***=FROM** *source_group* **THRU** *thru_point*[**THRU** *thru_point*] *delay***;** **TIMESPEC TS***id***=THRU** *thru_point*[**THRU** *thru_point*] **TO** *dest_group delay***;**

Spec Type	TS Attribute Syntax
Linked specification	Schematic syntax in TIMESPEC primitive: **TS***id***=FROM** *source_group* **TO** *dest_group another_TSid* [***** | /]*number* **TS***id***=FROM** *source_group another_TSid* [***** | /]*number* **TS***id***=TO** *dest_group another_TSid*[***** | /]*number* UCF syntax: **TIMESPEC TS***id***=FROM** *source_group* **TO** *dest_group another_TSid* [***** | /]*number***;** **TIMESPEC TS***id***=FROM** *source_group another_TSid* [***** | /]*number***;** **TIMESPEC TS***id***=TO** *dest_group another_TSid*[***** | /]*number***;**
Clock period	Schematic syntax in TIMESPEC primitive: **TS***id***=PERIOD** *TNM_reference period* {**HIGH** | **LOW**} [*high_or_low_time*] UCF syntax: **TIMESPEC TS***id***=PERIOD** *TNM_reference period* {**HIGH** | **LOW**} [*high_or_low_time*]**;**
Derived clocks	Schematic syntax in TIMESPEC primitive: **TS***id***=PERIOD** *TNM_reference another_PERIOD_identifier* [/ | *****]*number*{**HIGH** | **LOW**} [*high_or_low_time*] UCF syntax: **TIMESPEC TS***id***=PERIOD** *TNM_reference another_PERIOD_identifier* [/ | *****]*number*{**HIGH** | **LOW**} [*high_or_low_time*]**;**

Spec Type	TS Attribute Syntax
TS attribute priority	*normal_timespec_syntax* **PRIORITY** *integer*
Group OFFSETs	Schematic syntax in TIMESPEC primitive: **TS***identifier***=TIMEGRP** *name* **OFFSET=** {**IN** | **OUT**} *offset_time* [*units*] {**BEFORE** | **AFTER**} *clk_name* [**TIMEGRP** *group_name*] The UCF and PCF syntax do not require the TS*identifier*. UCF syntax: [**TIMEGRP** *name*] **OFFSET=** {**IN** | **OUT**} *offset_time* [*units*] {**BEFORE** | **AFTER**} *clk_name* [**TIMEGRP** *group_name*]**;** PCF syntax: [**TIMEGRP** *name*] **OFFSET=** {**IN** | **OUT**} *offset_time* [*units*] {**BEFORE** | **AFTER**} **COMP** *clk_iob_name* [**TIMEGRP** *group_name*]**;**

The following table lists additional attributes or constraints that are used in or affect TS attributes.

Attribute Syntax	Where Applied	How Used
Schematic syntax on net, pin, symbol, or macro:**TPTHRU=***identifier* UCF syntax: {**NET** | **PIN** | **INSTANCE**} *name* **TPTHRU=***identifier***;**	Net, symbol, pin, macro	In through point TS attribute
Schematic syntax on net, pin, symbol, or macro: **TPSYNC=***identifier* UCF syntax: {**NET** | **PIN** | **INSTANCE**} *name* **TPSYNC=***identifier***;**	Net, symbol, pin, macro	As group in TS attribute

Attribute Syntax	Where Applied	How Used
Schematic syntax on net or pin: **TIG** **TIG=***identifier* UCF syntax: {**NET** | **PIN**} *name* **TIG;** {**NET** | **PIN**} *name* **TIG=***identifier***;**	Net, pin	Prevents timing analysis
TS*identifier***=DROP_SPEC;** (Constraints file only)	N / A	Prevents timing analysis for **TS***identifier*

Other Constraints

The following table lists additional timing constraints.

Attribute Syntax	Where Applied	How Used
Schematic syntax on net or pin: **PERIOD** *period* {**HIGH**|**LOW**} [*high_or_low_time*] UCF syntax: {**NET** | **PIN**} *name* **PERIOD** *period* {**HIGH**|**LOW**} [*high_or_low_time*]**;**	Nets, pins	Specifies register clock period
Schematic syntax: **MAXSKEW=***allowable_skew* UCF syntax: **NET** *name* **MAXSKEW=***allowable_skew***;** PCF Syntax: {**NET** | **TIMEGRP** | **ALLCLOCKNETS**} *name* **MAXSKEW=***allowable_skew***;**	Nets, timegroups, ALLCLOCKNETS	Specifies skew

Attribute Syntax	Where Applied	How Used
Schematic syntax: **MAXDELAY=** *path_value* [**PRIORITY** *integer*] UCF syntax: **NET** *net_name* **MAXDELAY=** *path_value* [**PRIORITY** *integer*]; PCF syntax: **TS***identifier***=MAXDELAY** *path path_value* [**PRIORITY** *integer*]; {**NET** | **TIMEGRP** | **ALLCLOCKNETS**} *name* **MAXDELAY=** *path_value* [**PRIORITY** *integer*]; **PATH** *path_name* **MAXDELAY=** *path_value* [**PRIORITY** *integer*]; **ALLPATHS MAXDELAY=** *path_value* [**PRIORITY** *integer*]; **FROM** *group_item* **THRU** *group_item1... group_itemn* **MAXDELAY=** *path_value* [**PRIORITY** *integer*]; **FROM** *group_item* **THRU** *group_item1... group_itemn* **TO** *group_item* **MAXDELAY=** *path_value* [**PRIORITY** *integer*]; **THRU** *group_item1... group_itemn* **TO** *group_item* **MAXDELAY=** *path_value* [**PRIORITY** *integer*];	Nets, Paths, FROM THRU, FROM THRU TO, THRU TO	Specifies delay
Schematic syntax: **USELOWSKEWLINES** UCF syntax: **NET** *net_name* **USELOWSKEW-LINES;**	Nets	Routes nets to low skew lines

Chapter 7

Logical Design Rule Check

This program is compatible with the following families.

- Spartan / XL / -II
- Virtex / -E / -II
- XC9500 / XL / XV
- XC4000E / L / EX / XL / XV / XLA
- XC3000A / L
- XC3100A / L
- XC5200

This chapter describes the Logical Design Rule Check (DRC). The chapter contains the following sections.

- "Logical DRC"
- "Logical DRC Tests"

Logical DRC

The Logical DRC (Design Rule Check), is a series of tests run to verify the logical design in the NGD (Native Generic Database) file. The Logical DRC (also called the NGD DRC) performs device-independent checks; they do not depend on the FPGA to which you will eventually map the design.

The Logical DRC generates messages to show the status of the tests performed. Messages can be error messages (for conditions where the logic will not operate correctly) or warnings (for conditions where the logic is incomplete).

The Logical DRC runs automatically at the following times.

- At the end of NGDBuild, before NGDBuild writes out the NGD file

  NGDBuild writes out the NGD file if DRC warnings are discovered, but does not write out an NGD file if DRC errors are discovered.

- At the end of NGD2EDIF, NGD2VER, or NGD2VHDL, before writing out the netlist file

  The netlist writers do not perform the entire DRC. They only perform the Net and Name checks. A netlist writer writes out a netlist file even if DRC warnings or errors are discovered.

Logical DRC Tests

The Logical DRC performs six types of checks.

- Block check
- Net check
- Pad check
- Clock buffer check
- Name check
- Primitive pin check

The following sections describe these tests.

Block Check

The block check verifies that each terminal symbol in the NGD hierarchy (that is, each symbol that is not resolved to any lower-level components) is an NGD primitive. A block check failure is treated as an error. As part of the block check, the DRC also checks user-defined properties on symbols and the values on the properties to make sure they are legal.

Net Check

The net check determines the number of NGD primitive output pins (drivers), tristate pins (drivers) and input pins (loads) on each signal in the design. If a signal does not have at least one driver (or one tristate driver) and at least one load, a warning is generated. An error is generated if a signal has multiple non-tristate drivers or any combination of tristate and non-tristate drivers. As part of the net check, the DRC also checks user-defined properties on signals and the values on the properties to make sure they are legal.

Pad Check

The pad check verifies that each signal connected to pad primitives obeys the following rules.

- If the PAD is an input pad, the signal to which it is connected can only be connected to the following types of primitives.
  - Buffers
  - Clock buffers
  - PULLUP
  - PULLDOWN
  - BSCAN

  The input signal can be attached to multiple primitives, but only one of each of the above types. For example, the signal can be connected to a buffer primitive, a clock buffer primitive, and a PULLUP primitive, but it cannot be connected to a buffer primitive and two clock buffer primitives. Also, the signal cannot be connected to both a PULLUP primitive and a PULLDOWN primitive. Any violation of the rules above results in an error, with the exception of signals attached to multiple pull-ups or pull-downs, which produces a warning. A signal which is not attached to any of the above types of primitives also produces a warning.
- If the PAD is an output pad, the signal it is attached to can only be connected to one of the following primitive outputs.
  - A single buffer primitive output
  - A single tristate primitive output
  - A single BSCAN primitive

In addition, the signal can also be connected to one of the following primitives.

- A single PULLUP primitive
- A single PULLDOWN primitive

Any other primitive output connections on the signal results in an error.

If the condition above is met, the output PAD signal may also be connected to one clock buffer primitive *input*, one buffer primitive *input*, or both.

- If the PAD is a bidirectional or unbonded pad, the signal it is attached to must obey the rules stated above for input and output pads. Any other primitive connections on the signal results in an error. The signal connected to the pad must be configured as both an input and an output signal; if it is not, you receive a warning.
- If the signal attached to the pad has a connection to a top-level symbol of the design, that top-level symbol pin must have the same type as the pad pin, except that output pads can be associated with tristate top-level pins. A violation of this rule results in a warning.
- If a signal is connected to multiple pads, an error is generated. If a signal is connected to multiple top-level pins, a warning is generated.

Clock Buffer Check

The clock buffer configuration check verifies that the output of each clock buffer primitive is connected to only inverter, flip-flop or latch primitive clock inputs, or other clock buffer inputs. Violations are treated as warnings.

Name Check

The name check verifies the uniqueness of names on NGD objects as defined below. The tests, and the messages reported by a violation of the tests, are as follows.

- Pin names must be unique within a symbol. A violation results in an error.

- Instance names must be unique within the instance's position in the hierarchy (that is, a symbol cannot have two symbols with the same name under it). A violation results in a warning.
- Signal names must be unique within the signal's hierarchical level (that is, if you push down into a symbol, you cannot have two signals with the same name). A violation results in a warning.
- Global signal names must be unique within the design. A violation results in a warning.

Primitive Pin Check

The primitive pin check verifies that certain pins on certain primitives are connected to signals in the design. The following table shows which pins are tested on each NGD primitive type.

Table 7-1 Checked Primitive Pins

NGD Primitive	Pins Checked
X_TRI	IN, OUT, and CTL
X_FF	IN, OUT, and CLK
X_LATCH	IN, OUT, and CLK
X_IPAD	PAD
X_OPAD	PAD
X_BPAD	PAD

Note If one of these pins is not connected to a signal, you receive a warning.

Chapter 8

MAP—The Technology Mapper

This program is compatible with the following families.

- Spartan / -II / XL
- Virtex / -E / -II
- XC3000A / L
- XC3100A / L
- XC4000E / L
- XC4000EX / XL / XV / XLA
- XC5200

This chapter describes MAP. The chapter contains the following sections.

- “MAP”
- “MAP Syntax”
- “MAP Files”
- “MAP Options”
- “The MAP Process”
- “Register Ordering”
- “Guided Mapping”
- “Simulating Map Results”
- “The MAP Report (MRP) File”
- “Halting MAP”

MAP

MAP maps a logical design to a Xilinx FPGA. The input to mapping 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, and any number of NMC (macro library) files, each of which contains the definition of a physical macro. MAP first performs a logical DRC (Design Rule Check) on the design in the NGD file. MAP then maps the logic to the components (logic cells, I/O cells, and other components) in the target Xilinx FPGA. The output design is an NCD (Native Circuit Description) file – a physical representation of the design mapped to the components in the Xilinx FPGA. The NCD file can then be placed and routed.

The flow through MAP is shown in the following figure. MAP can be invoked from the Design Manager/Flow Engine graphical interface or from the UNIX or DOS command line. The Design Manager/Flow Engine is described in the *Design Manager/Flow Engine Guide*. This chapter describes running MAP from the UNIX or DOS command line.

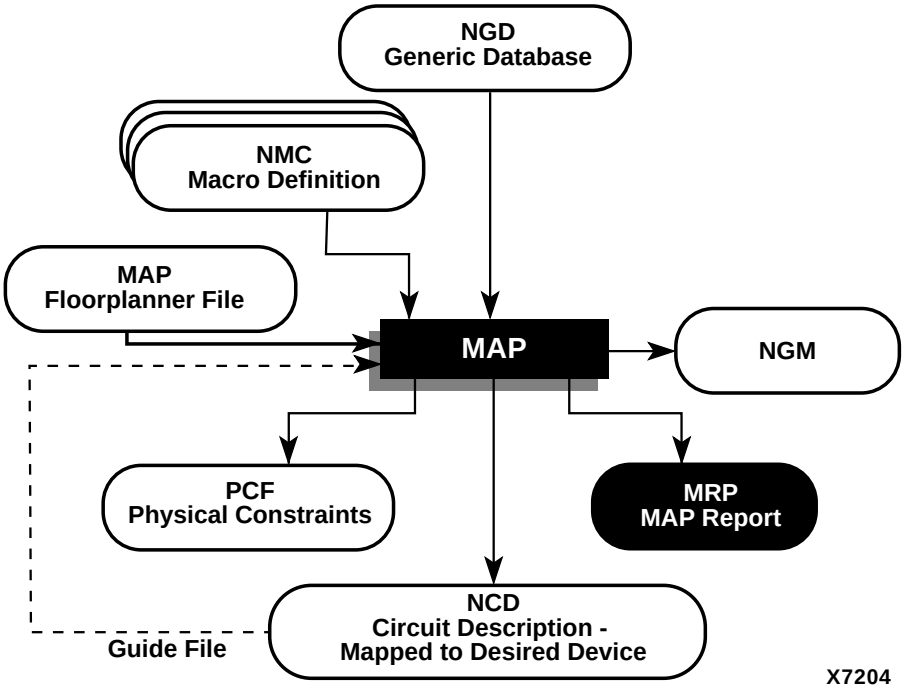


Figure 8-1 MAP

MAP Syntax

The following syntax maps your design.

map [*options*] *infile*[**.ngd**] [*pcf_file*[**.pcf**]]

Options can be any number of the MAP options listed in the "MAP Options" section. They do not need to be listed in any particular order. Separate multiple options with spaces.

Infile[**.ngd**] is the input NGD file. You do not have to enter the .ngd extension.

Pcf_file[**.pcf**] is the Physical Constraints File in PCF format. A constraints file name is optional on the command line, but if one is entered it must be entered after the input file name. You do not have to enter the .pcf extension. The constraints file name and its location are determined in this way.

- If you do not specify a physical constraints file name on the command line, the physical constraints file has the same name as the output file, with a .pcf extension. The file is placed in the output file's directory.
- If you specify a physical constraints file with no path specifier (for example, **cpu_1.pcf** instead of **/home/designs/cpu_1.pcf**), the .pcf file is placed in the current working directory.
- If you specify a physical constraints file name with a full path specifier (for example, **/home/designs/cpu_1.pcf**), the physical constraints file is placed in the specified directory.
- If the physical constraints file already exists, MAP reads the file, checks it for syntax errors, and overwrites the schematic-generated section of the file. MAP also checks the user-generated section for errors and corrects errors by commenting out physical constraints in the file or by halting the operation. If no errors are found in the user-generated section, the section remains the same.

For a discussion of the output file name and its location, see the "–o (Output File Name)" section.

MAP Files

This section describes the MAP input and output files.

Input Files

MAP uses the following files as inputs.

- NGD file—Native Generic Database file. This 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. The file also contains all of the constraints applied to the design during design entry or entered in a UCF (User Constraints File). The NGD file is created by NGDBuild.

- NMC files—Macro library files. An NMC file contains the definition of a physical macro. When there are macro instances in the NGD design file, NMC files are used to define the macro instances. There is one NMC file for each *type* of macro in the design file.
- Guide NCD file—An optional input file generated from a previous MAP run. An NCD file contains a physical description of the design in terms of the components in the target Xilinx device. A guide NCD file is an output NCD file from a previous MAP run that is used as an input to guide a later MAP run.
- Guide NGM file—A binary design file containing all of the data in the input NGD file as well as information on the physical design produced by the mapping. See "Guided Mapping" section for details.
- MFP—Map Floorplanner File, which is generated by the Floorplanner, specified as an input file with the -fp option. The MFP file is essentially used as a guide file for mapping. To create a Map Floorplanner File, you must first have generated an NGD file and a mapped NCD file. When you have run MAP to generate an NCD file, you can open the mapped NCD file in the Floorplanner, modify the placement of components, and then generate an MFP file. You can then use the MFP file as an input file with the -fp map option. The MFP file is only created for the XC4000, Spartan/-II, and Virtex/-E/-II architectures.
- MDF file—MAP Directive File. The MDF is an optional input file used for guided mapping. The MDF file describes how logic was decomposed when the guide design was mapped. MAP uses the hints in the MDF as a guide for logic decomposition in the guided mapping run.

Output Files

Output from MAP consists of the following files.

- NCD file—Native Circuit Description. A physical description of the design in terms of the components in the target Xilinx device. For a discussion of the output NCD file name and its location, see the "–o (Output File Name)" section.
- PCF (Physical Constraints) file—an ASCII text file containing the constraints specified during design entry expressed in terms of

physical elements. The physical constraints in the PCF file are expressed in Xilinx's constraint language. This file is described in "Physical Constraints (PCF) File" chapter.

MAP either creates a PCF file if none exists or rewrites an existing file by overwriting the schematic-generated section of the file (between the statements SCHEMATIC START and SCHEMATIC END). For an existing physical constraints file, MAP also checks the user-generated section for syntax errors, and signals errors by halting the operation. If no errors are found in the user-generated section, the section is unchanged.

- NGM file—a binary design file containing all of the data in the input NGD file as well as information on the physical design produced by the mapping. The NGM file is used to correlate the back-annotated design netlist to the structure and naming of the source design.
- MRP (MAP report) file—a file containing information about the MAP command run. The MRP file lists any errors and warnings found in the design, lists design attributes specified, details on how the design was mapped (for example, the logic that was removed or added and how signals and symbols in the logical design were mapped into signals and components in the physical design). The file also supplies statistics about component usage in the mapped design. See "The MAP Report (MRP) File" section for more details.
- MDF (MAP Directive File)—a file describing how logic was decomposed when the design was mapped. In guided mapping, MAP uses the hints in the MDF as a guide for logic decomposition.

The MRP, MDF and NGM files produced by a MAP run all have the same name as the output file, with the appropriate extension. If the MRP, MDF or NGM files already exist, they are overwritten by the new files.

MAP Options

The following table illustrates which architectures can be used with each option.

Table 8-1 Map Options and Architectures

Options	Architectures
-b	Spartan, xc4000e / l
-c	Spartan, SpartanXL, Spartan-II, xc3000a / l, xc3100 / a / l, xc4000e / ex / l / xl / xla / xv, xc5200, Virtex-E / -II
-cm	Spartan, SpartanXL, Spartan-II, xc3000a / l, xc3100 / a / l, xc4000e / ex / l / xl / xla / xv, xc5200, Virtex / -E / -II,
-d	xc3000a / l, xc3100a / l
-detail	all FPGA architectures
-f	Spartan, SpartanXL, Spartan-II, Virtex / -E / -II, xc3000a / l, xc3100 / a / l, xc4000e / ex / l / xl / xla / xv, xc5200
-fp	Spartan, SpartanXL, Spartan-II, xc4000e / ex / l / xl / xla / xv, xc5200, Virtex / -E / -II
-gf	Spartan, SpartanXL, xc3000a / l, xc3100 / a / l, xc4000e / ex / l / xl / xla / xv, xc5200
-gm	Spartan, SpartanXL, xc3000a / l, xc3100 / a / l, xc4000e / ex / l / xl / xla / xv, xc5200
-ir	Spartan, SpartanXL, Spartan-II, Virtex / -E / -II, xc4000e / ex / l / xl / xla / xv, xc5200
-k	Spartan, SpartanXL, Spartan-II, Virtex / -E / -II, xc4000e / ex / l / xl / xla / xv, xc5200
-l	Spartan, SpartanXL, Spartan-II, Virtex / -E / -II, xc4000e / ex / l / xl / xla / xv, xc5200
-o	All architectures
-oe	Spartan, SpartanXL, xc3000a / l, xc3100 / a / l, xc4000e / ex / l / xl / xla / xv, xc5200
-os	Spartan, SpartanXL, xc3000a / l, xc3100 / a / l, xc4000e / ex / l / xl / xla / xv, xc5200
-p	Spartan, SpartanXL, Spartan-II, Virtex / -E / -II, xc3000a / l, xc3100 / a / l, xc4000e / ex / l / xl / xla / xv, xc5200

Table 8-1 Map Options and Architectures

Options	Architectures
-pr	Spartan, SpartanXL, Spartan-II, Virtex / -E / -II, xc3000a / l, xc3100 / a / l, xc4000e / ex / l / xl / xla / xv
-r	Spartan, SpartanXL, Spartan-II, xc3000a / l, xc3100 / a / l, xc4000e / ex / l / xl / xla / xv, Virtex / -E / -II
-tx	Spartan-II, Virtex / -E / -II
-u	Spartan, SpartanXL, Spartan-II, Virtex / -E / -II, xc3000a / l, xc3100 / a / l, xc4000e / ex / l / xl / xla / xv, xc5200

–b (Convert Clock Buffers—XC4000E/L and Spartan Only)

Note This option does not apply to the Spartan-II or SpartanXL architecture.

The –b option replaces GCLKs and ACLKs (primary and secondary clocks) with a generic clock buffer (CKBUF) prior to mapping. This option is useful when you are mapping an XNF netlist created in the Synopsys environment where all clocks are mapped to BUFGP (primary clock buffers) and secondary clocks are not used. The –b option gives MAP the greatest amount of latitude in choosing the clock assignments.

Note MAP does not override the LOC= constraint.

–c (Pack CLBs)

–c [*packfactor*]

The –c option determines the degree to which CLBs are packed when the design is mapped. The valid range of values for the *packfactor* is 0–100.

The *packfactor* values ranging from 1 to 100 roughly specify the percentage of CLBs available in a target device for packing your design's logic.

A *packfactor* of 100 means that all CLBs in a target part are available for design logic. A *packfactor* of 100 results in minimum packing density, while a *packfactor* of 1 represents maximum packing density. Specifying a lower *packfactor* results in a denser design, but the design may then be more difficult to place and route.

The **–c 0** option specifies that only *related* logic (that is, logic having signals in common) should be packed into a single CLB. Specifying **–c 0** yields the least densely packed design.

For values of –c from 1 to 100, MAP merges unrelated logic into the same CLB only if the design requires more resources than are available in the target device (an "overmapped" design). If there are *more* resources available in the target device than are needed by your design, the number of CLBs utilized when **–c 100** is specified may equal the number required when **–c 0** is specified.

Note The **–c 1** setting should only used to determine the maximum density (minimum area) to which a design can be packed; it should almost never be used in the actual implementation of a design. Designs packed to this maximum density generally have longer run times, severe routing congestion problems in PAR, and poor design performance.

The default *packfactor* (the value if you do not specify a –c option, or enter a –c option without a *packfactor*) is 97% for the XC4000E architecture and 100% for all other XC4000 architectures, Virtex / -E / -II, and Spartan / XL / -II.

Processing a design with the **–c 0** option is a good way to get a first estimate of the number of CLBs required by your design.

–cm (Cover Mode)

```
-cm {area | speed | balanced}
```

The –cm option specifies the criteria used during the "cover" phase of MAP. In the "cover" phase, MAP assigns the logic to CLB function generators (LUTs).

- The **area** setting makes reducing the number of LUTs (and therefore the number of CLBs) the highest priority.
- The **speed** setting makes reducing the number of *levels* of LUTS (the number of LUTs a path passes through) the highest priority. This setting makes it easiest to achieve your timing constraints after the design is placed and routed. For most designs there is a small increase in the number of LUTs (compared to the **area** setting), but in some cases the increase may be large.
- The **balanced** setting balances the two priorities—reducing the number of LUTs and reducing the number of levels of LUTs. It produces results similar to the **speed** setting but avoids the possibility of a large increase in the number of LUTs.

The default setting for the –cm option is **area** (cover for minimum number of LUTs). For synthesis based designs, changing the default does not result in improved performance.

–d (Use DI Pin—XC3000 Architectures Only)

If you specify this option, MAP can use the DI (Direct Input) pin of each CLB in the device for the XC3000A, XC3000L, XC3100A and XC3100L architectures. If you use this pin, the setup time requirement for each CLB flip-flop is reduced, but the DI pin has a hold time requirement (which none of the other CLB logic input pins has). Using the DI pin results in a denser design, but the design may then be more difficult to place and route. Even if you specify the –d option, MAP tries to minimize the use of the DI pin.

–detail (Write Out Detailed MAP Report)

This option writes out a detailed MAP report. The option replaces the MAP_REPORT_DETAIL environment variable.

–f (Execute Commands File)

-f *command_file*

The –f option executes the command line arguments in the specified *command_file*. For more information on the –f option, see the "–f Option" section of the "Introduction" chapter.

–fp (Floorplanner)

-fp *filename*.**mfp**

The –fp option requires the specification of an existing MFP file created by the Floorplanner. The MFP file is essentially used as a guide file for mapping.

The MFP file is created in the Floorplanner from a previously mapped NCD file. If you use the -fp option, you cannot use the guide file option (-gf).

The -fp option can be used with the XC4000/E/L, XC4000EX/XL/XLA/XV, Spartan/XL/-II, and Virtex/-E/-II architectures.

For more information about the Floorplanner, see the *Floorplanner Reference/User Guide*.

–gf (Guide NCD File)

-gf *guidefile*

The –gf option specifies the name of an existing NCD file (from a previous MAP run) to be used as a guide for the current MAP run. For Virtex/-E/-II architectures, guided mapping also uses the NGM file. For a description of guided mapping, see the "Guided Mapping" section.

–gm (Guide Mode)

-gm {**exact** | **leverage**}

The –gm option specifies the form of guided mapping to be used**.**

In the EXACT mode the mapping in the guide file is followed exactly. In the LEVERAGE mode, the guide design is used as a starting point for mapping but, in cases where the guided design tools cannot find matches between net and block names in the input and guide designs, or your constraints rule out any matches, the logic is not guided.

For a description of guided mapping, see the "Guided Mapping" section.

–ir (Do Not Use RLOCs to Generate RPMs)

If you enter the –ir option, MAP uses RLOC constraints to group logic within CLBs, but does not use the constraints to generate RPMs (Relationally Placed Macros) controlling the relative placement of CLBs. Stated another way, the RLOCs are not used to control the relative placement of the CLBs with respect to each other.

For the XC4000 and Spartan architectures, the -ir option has an additional behavior; the RLOC constraint that cannot be met is ignored and the mapper will continue processing the design. A warning is generated for each RLOC that is ignored. The resulting mapped design is a valid design.

This option does not apply to the XC3000 architecture.

–k (Map to Input Functions)

The syntax for Spartan-II and Virtex/-E/ architectures follows:

–k {4 |5 |6}

The syntax for the Virtex-II architecture follows:

–k {4 |5 |6| *7*| *8*}

You can specify the maximum size function that is covered. The default is 4. For compatibility with M1.5, use –k 4. Covering to 5, 6, 7 or 8 input functions results in the use of F5MUX, F6MUX, and FXMUX.

For the XC4000 and XC5200 architectures, only –k is specified (not 4, 5 or 6). If the –k option is specified, logic functions of five inputs are mapped into a single CLB (if possible). To perform this mapping, all three of the function generators in the CLB may be used

By mapping input functions into single CLBs, the –k option may produce a mapping with fewer levels of logic, thus eliminating a number of CLB-to-CLB delays. On the other hand, using the –k option may prevent logic from being packed into CLBs in a way that minimizes CLB utilization.

For synthesis-based designs, specifying **–k 4** has litle effect. This is because MAP combines smaller input functions into large functions such as F5MUX, F6MUX, F7MUX and F8MUX.

The –k option does not apply to the XC3000 architecture.

–l (No logic replication)

If you do not specify the –l option, MAP can perform logic replication, a logic optimization in which the program takes a single driver driving multiple loads and maps it as multiple components, each driving a single load (see the following figure). Logic replication results in a mapping that often makes it easier to meet your timing requirements, since some delays can be eliminated on critical nets.

This option does not apply to the XC3000 and XC5200 architectures.

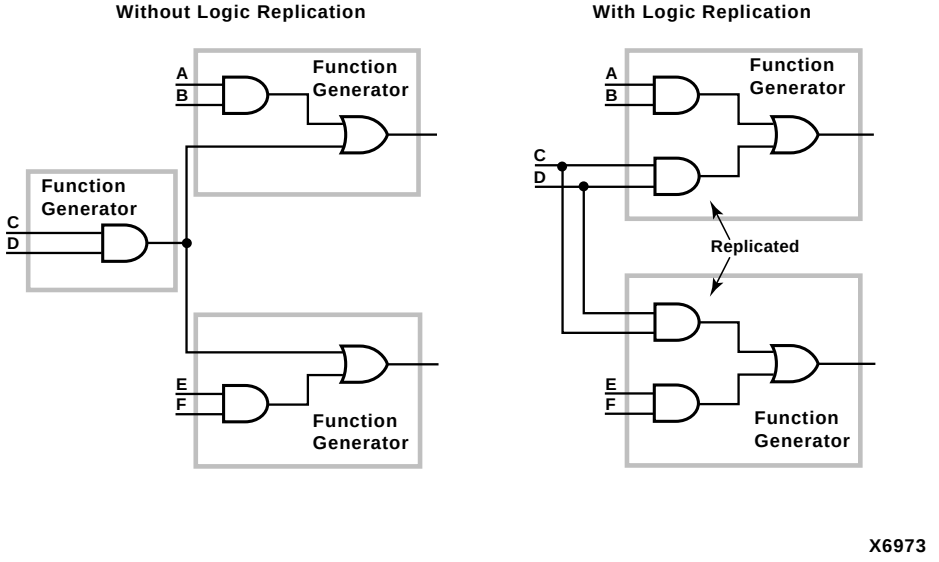


Figure 8-2 Logic Replication (–l Option)

–o (Output File Name)

–o *outfile*[**.ncd**]

Specifies the name of the output NCD file for the design. The .ncd extension is optional. The output file name and its location are determined in this way.

- If you do not specify an output file name with the –o option, the output file has the same name as the input file, with an .ncd extension. The file is placed in the input file's directory
- If you specify an output file name with no path specifier (for example, **cpu_dec.ncd** instead of **/home/designs/cpu_dec.ncd)**, the NCD file is placed in the current working directory.
- If you specify an output file name with a full path specifier (for example, **/home/designs/cpu_dec.ncd**), the output file is placed in the specified directory.

If the output file already exists, it is overwritten with the new NCD file. You do not receive a warning when the file is overwritten.

–oe (Logic Optimization Effort)

-oe {**normal** | **high**}

The –oe option specifies the effort MAP uses when performing logic optimization. In the **high** setting, MAP exerts a greater effort to optimize combinatorial logic, but the mapping takes longer to complete. The **high** setting must be used if the input to the MAP is not optimized, for example, a design created in XABEL.

For the –oe option to apply, the –os (logic optimization style) option must be enabled; that is, –os must have a setting other than **none.**

If logic optimization is specified by the –os option, the default setting for the –oe option is **normal**.

See the following section for guidelines on when to use logic optimization.

Note Logic optimization has little effect on most synthesis-based designs.

This option does not apply to Virtex/-E/-II or Spartan-II.

–os (Logic Optimization Style)

-os {**area** | **speed** | **balanced**}

Logic optimization in the context of MAP refers to FPGA-specific 4-input lookup optimization by the OPTIX optimizer.

The –os option specifies what type of logic optimization MAP performs.

- The **area** setting optimizes combinatorial logic in a way that uses the minimum number of logic cell function generators (LUTs). This setting minimizes the amount of device area taken up when the design is placed and routed.
- The **speed** setting optimizes in a way that makes it easiest to achieve your timing constraints after the design is placed and routed, even if more function generators must be used.
- The **balanced** setting gives you the optimum combination of area and speed.

The default setting for the –os option disables logic optimization—no optimization is performed. You may want to avoid performing logic optimization in the following cases.

- Your design has already been optimized (for example, a design created in the Synopsys toolset). If the input to MAP has already been optimized (including FPGA-specific 4-input LUT optimization), the MAP results with the –os option enabled may be worse than without the option.
- Your design has been entered as a schematic using Xilinx Unified Library components. In this case, the MAP results with the –os option enabled may be worse than without the option.
- You want to make sure you can perform back-annotation on any of the logic in your original design. Optimization may make some of the logic unavailable for back-annotation.

Note: After combinatorial logic optimization has been performed, you lose the correlation between signal names in the NCD file and signal names in the original design. User signal names are not preserved within optimized combinatorial networks. This affects back-annotation and also results in a reduction in the amount of guided mapping and guided placement and routing that can be performed. However, signals connected to pads or to the outputs of tbufs, flip-flops, latches, and RAMS are preserved for back-annotation.

This option does not apply to Virtex/-E/-II or Spartan-II.

–p (Xilinx Part Number)

–p *part*

Specifies the Xilinx part number for the device. The syntax for the –p option is described in the "Part Numbers in Commands" section of the "Introduction" chapter. Examples of *part* entries are **XC4003E-PC84**, and **XC4028EX-HQ240-3**.

If you do not specify a part number using the –p option, MAP selects the part specified in the input NGD file. If the information in the input NGD file does not specify a complete device and package, you must enter a device and package specification using the –p option. MAP supplies a default speed value, if necessary.

Note: The architecture you specify with the –p option must match the architecture specified within the input NGD file.
You may have chosen this architecture when you ran NGDBuild or during an earlier step in the design entry process (for example, you may have specified the architecture as an attribute within a schematic, or specified it as an option to a netlist reader). If the architecture does not match, you have to run NGDBuild again and specify the desired architecture.

Note You can only enter a part number or device name from a device library you have installed on your system. For example, if you have not installed the 4006E device library, you cannot create a design using the 4006E–PC84 part.

–pr (Pack Registers in I/O)

–pr {**i** | **o** | **b**}

When MAP runs without the –pr option, MAP can only place flip-flops or latches within an I/O cell if your design entry method specifies that these components are to be placed within I/O cells. For example, if you create a schematic using IFDX (Input D Flip-Flop) or OFDX (Output D Flip-Flop) design elements, the physical components corresponding to these design elements must be placed in I/O cells. The –pr option specifies that flip-flops or latches may be packed into input registers (**i** selection), output registers (**o** selection), or both (**b** selection) even if the components have not been specified in this way. This option does not apply to the XC5200 architecture.

–r (No Register Ordering)

The –r option disables register ordering. If you specify this option, register bit names are ignored when registers are mapped, and the bits are not mapped in any special order. If you do not specify this option, MAP looks at the register bit names for similarities and tries to map register bits in an ordered manner (called “register ordering”). For a description of register ordering, see the “Register Ordering” section.

Note For Virtex/-E/-II, If you are migrating from an M1.5 design to a 3.1i design, you must use the -r option to maintain the behavior of the M1.5 design.

This option does not apply to the XC5200 architecture.

-tx (Transform Buses)

-tx {**on** | **off** | **aggressive** | **limit**}

The -tx option specifes what type of bus transformation MAP performs. The four permitted settings are on, off, aggressive, and limit. The following example illustrates how the settings are used.

This example design is mapped to a XCV50 Virtex device.

Bus A has 4 TBUFs

Bus B has 20 TBUFs

Bus C has 30 TBUFs

The default behavior in the MAP program is as follows:

- The **on** setting performs partial transformation for a long chain that exceeds the device limit.

  Bus A is transformed to LUTs (number of TBUFs is >1, ≤4)

  Bus B is transformed to CY chain (number of TBUFs is >4, ≤48)

  Bus C is *partially* transformed. (25 TBUF + 1 dummy TBUF due to the maximum width of the XCV50 device + CY chain implementing the other 5 TBUFS)

- The **off** setting turns bus transformation off. This is the default setting.

- The **aggressive** setting transforms the entire bus.

  Buses A, B have the same result as the *on* setting.

  Bus C is implemented entirely by CY chain. (30 ≤ the default upper limit for carry chain transformation)

- The **limit** setting is the most conservative. It transforms only that portion of the bus that exceeds the device limit.

–u (Do Not Remove Unused Logic)

If –u is specified, MAP maps unused components and nets in the input design and includes them as part of the output design. If –u is not specified, MAP eliminates unused components and nets from the design before mapping.

The –u option is helpful if you want to run a preliminary mapping on an unfinished design, possibly to see how many components the mapped design uses. By specifying –u, you are assured that all of the design's logic (even logic that is part of incomplete nets) is mapped.

The MAP Process

To map a design, MAP performs these steps.

1. Choose the target Xilinx device, package, and speed. MAP selects a part in this way.
   - If a part is specified on the MAP command line, this is the part used.
   - If the command line does not specify a part, MAP selects the part specified in the input NGD file. If the information in the input NGD file does not specify a complete architecture, device, and package, you receive an error message and MAP does not continue. MAP supplies a default speed if necessary
2. Read the information in the input design file.
3. Perform 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 the MAP run continues. The Logical DRC (also called the NGD DRC) is described in "Logical Design Rule Check" chapter.

Note: Step 3 is skipped if the NGDBuild DRC was successful.

4. Assign the device global clock buffers (if possible).
5. Remove unused logic. All unused components and nets are removed, unless these conditions exist.
   - A Xilinx S (Save) constraint has been placed on a net during design entry. If an unused net has an S constraint, the net and all used logic connected to the net (as drivers or loads) is retained. All unused logic connected to the net is deleted. For a more complete description of the S constraint, see the "Attributes, Constraints, and Carry Logic" chapter of the *Libraries Guide.*
   - The –u option was specified on the MAP command line. If this option is specified, all unused logic is kept in the design.

6. Map pads and their associated logic into IOBs.
7. Map the logic into Xilinx components (IOBs, CLBs, etc.). If any Xilinx mapping control symbols appear in the design hierarchy of the input file (for example, FMAP or HMAP symbols targeted to an XC4000EX device), MAP uses the existing mapping of these components in preference to remapping them. The mapping is influenced by various constraints; these constraints are described in the "Attributes, Constraints, and Carry Logic" chapter of the *Libraries Guide.*
8. Update the information received from the input NGD file and write 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 On Virtex/-E/-II devices, guided mapping uses the NGM file. For more information, see the "Guided Mapping" section.
9. Create a physical constraints (PCF) file. This is a text file containing 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 or indirectly through the FPGA Editor.

   MAP either creates a PCF file if none exists or rewrites an existing file by overwriting the schematic-generated section of the file (between the statements SCHEMATIC START and SCHEMATIC END). For an existing constraints file, MAP also checks the user-generated section and may either comment out constraints with errors or halt the program. If no errors are found in the user-generated section, the section remains the same.

   Note On a Virtex/-E/-II design, you must use a MAP generated PCF file . The timing tools perform skew checking only with a MAP0-generated PCF file.
10. Create an MDF file, which describes how logic was decomposed when the design was mapped. The MDF file is used for guided mapping.

    This step does not apply to Virtex/-E/-II or Spartan-II.
11. Run a physical Design Rule Check (DRC) on the mapped design. If DRC errors are found, MAP does not write an NCD file.

12. Create an NCD file, which represents the physical design. The NCD file describes the design in terms of Xilinx components—CLBs, IOBs, etc.
13. Write 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.

Register Ordering

When you map a design containing registers, the MAP software can optimize the way the registers are grouped into CLBs (slices for Virtex/-E/-II or Spartan-II—there are two slices per CLB). This optimized mapping is called "register ordering."

A CLB (Virtex/-E/-II or Spartan-II slice) has two flip-flops, so two register bits can be mapped into one CLB. For PAR (Place And Route) to place a register in the most effective way, you often want as many pairs of contiguous bits as possible to be mapped together into the same CLBs (for example, bit 0 and bit 1 together in one CLB, bit 2 and bit 3 in another).

MAP pairs register bits (performing "register ordering") if it can recognize that a series of flip-flops comprise a register. When you create your design, you can name register bits in a way that ensures they are mapped using register ordering.

Note MAP *does not* perform register ordering on any flip-flops which have BLKNM, LOC, or RLOC properties attached to them. The BLKNM, LOC, and RLOC properties define how blocks are to be mapped, and these properties override register ordering.

To be recognized as a candidate for register ordering, the flip-flops must have these characteristics.

- The flip-flops must share a common clock signal and common control signals (for example, Reset and Clock Enable).
- The flip-flop output signals must all be named according to this convention.

- Output signal names must begin with a common root containing at least one alphabetic character.
- The names must end with numeric characters or with numeric characters surrounded by parentheses (“(“ and “)”), angle brackets (“<“ and “>”), or square brackets (“[“ and “]”).

  For example, acceptable output signal names for register ordering are as follows.

data1	addr(04)	bus<1>
data2	addr(08)	bus<2>
data3	addr(12)	bus<3>
data4	addr(16)	bus<4>
		bus<5>

If a series of flip-flops is recognized as a candidate for register ordering, they are paired in CLBs in sequential numerical order. For example, in the first set of names shown above, data1 and data2, are paired in one CLB, while data3 and data4 are paired in another.

In the example below, no register ordering is performed, since the root names for the signals are not identical

data01

addr02

atod03

dtoa04

Note In the OrCAD® schematic capture program, an underbar (_) and a sheet number are appended to each output signal name (for example, data01_1 or add15_12). In order to allow register ordering on designs developed using the OrCAD tools, MAP checks each signal name to see if it ends with an underbar followed by numeric characters.

When it finds a signal with this type of name, MAP ignores the underbar and the numeric characters when it considers the signal for register ordering. For example, if signals are named data00_1 and data01_2, MAP considers them as data00 and data01 for purposes of register ordering. These two signals *are* mapped to the same CLB.

Two extra notes:

- MAP does not change signal names when it checks for underbars—it only ignores the underbar and the number when it checks to see if the signal is a candidate for register ordering.
- Because of the way signals are checked, make sure you don't use an underbar as your bus delimiter. If you name a bus signal data0_01 and a non-bus signal data1, MAP sees them as data0 and data1 and register orders them even though you do not want them register ordered.

When you run the MAP command the default setting performs register ordering. If you specify the –r option when you run the command, the software does not perform register ordering and maps the register bits as if they were unrelated.

Guided Mapping

In guided mapping, an existing NCD is used to guide the current MAP run. The guide file may be from any stage of implementation: unplaced or placed, unrouted or routed. We recommend that you generate your NCD file using the current release of Xilinx software. However, MAP does support guided mapping using NCD files from the previous release.

The following figure shows the flow used when you perform guided mapping.

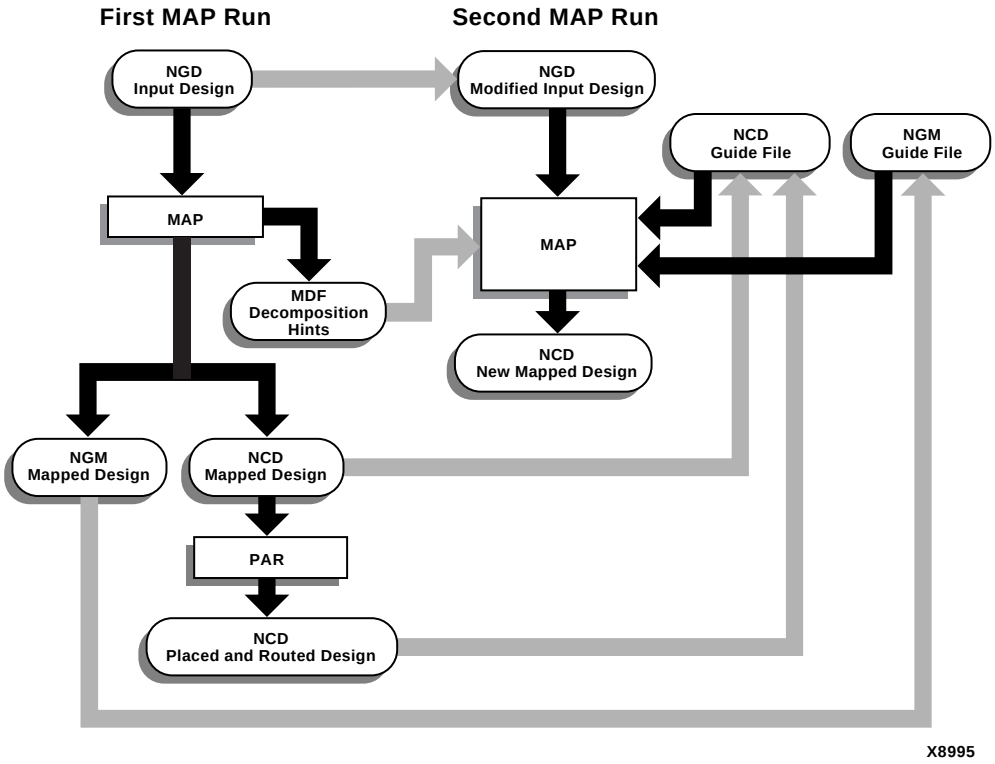


Figure 8-3 Guided Mapping

In the EXACT mode the mapping in the guide file is followed exactly. Any logic in the input NGD file that matches logic mapped into the physical components of the NCD guide file is implemented exactly as in the guide file. Mapping (including signal to pin assignments), placement and routing are all identical. Logic that is not matched to any guide component is mapped by a subsequent mapping step.

If there is a match in EXACT mode, but your constraints would conflict with the mapping in the guide file component, an error is posted. If an error is posted, you can modify the constraints to eliminate conflicts, change to the LEVERAGE guide mode (which is less restrictive), modify the logical design changes to avoid conflicts, or abandon using guided design.

In the LEVERAGE mode, the guide design is used as a starting point in order to speed up the design process. However, in cases where the guided design tools cannot find matches or your constraints rule out any matches, the logic is not guided. Whenever the guide design conflicts with the your mapping, placement or routing constraints, the guide is ignored and your constraints are followed.

Since the LEVERAGE mode only uses the guide design as a starting point for mapping, MAP may subsequently choose to alter the mapping to improve the speed or density of the implementation (for example, MAP may choose to collapse additional gates into a guided CLB).

For Spartan and Virtex/-E/-II devices, MAP uses the NGM as well as the NCD files as guides. You do not additionally specify the NGM file on the command line; MAP infers the appropriate NGM file from the NCD file you specify. If MAP does not find an NGM file in the same directory as the NCD, it generates a warning. In this case, MAP uses the NCD file alone as the guide file.

Note: For Verilog® or VHDL netlist input designs, re-synthesizing modules typically cause signal and instance names in the resulting netlist to be significantly different from the netlist obtained in earlier synthesis runs. This occurs even if the source level Verilog or VHDL code only contains a small change. Because guided mapping depends on signal and component names, synthesis designs often have a low "match rate" when guided. Therefore, guided mapping is not recommended for most synthesis-based designs, although there may be cases where it could be a successful alternative technique.

Simulating Map Results

When simulating from NGM files, you are not simulating a mapped result, that is, you are simulating the logical circuit description. Simulating from the NCD file actually simulates the physical circuit description.

MAP may generate an error that is not detected in the back-annotated simulation netlist. For example, you may run the following command after running MAP to generate the backannotated simulation netlist.

```
ngdanno mapped.ncd mapped.ngm -o mapped.nga
```

This command creates a back-annotated simulation netlist using the logical-to-physical cross-reference file named mapped.ngm. This

cross-reference file contains information about the logical design netlist which means that the back-annotated simulation netlist (mapped.nga) is actually a back-annotated version of the logical design. However, if MAP makes a physical error, for example, implements an Active Low function for an Active High function, this error will not be detected in the mapped.nga file which means that the error will not show up in the simulation.

Consider the following logical circuit generated by NGDBuild from an input design file.

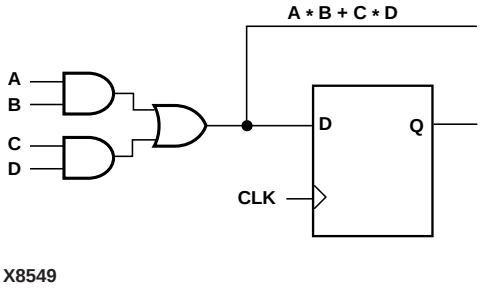


Figure 8-4 Logical Circuit Representation

Note the Boolean output from the combinatorial logic. Suppose that after running MAP for the preceding circuit, you obtain the following result.

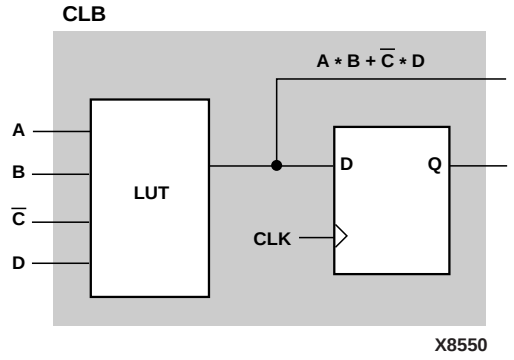


Figure 8-5 CLB Configuration

Note that MAP has generated an active low (C) instead of an active high (C). Therefore, the Boolean output for the combinatorial logic is incorrect. When you run NGDAnno using the mapped.ngm file (ngdanno mapped.ncd mapped.ngm -o mapped.nga), you will not detect this logical error because the delays are back-annotated to the correct logical design not to the physical design.

One way to detect the error is by running the NGDAnno command without using the mapped.ngm cross-reference file.

```
ngdanno mapped.ncd -o mapped.nga
```

Then physical simulations using the mapped.nga file will normally detect a physical error. However, even though, an error is detected, the specific type of error is not easily recognizable. You can use the FPGA Editor to try to pinpoint the error or call Xilinx Customer Support. It is also possible that the physical simulation is reporting an error that does not exist, that is, the CLB configuration is correct. In that instance, you can use the FPGA Editor to determine if the CLB is correctly modelled.

Lastly, if both the logical and physical simulations do not discover existing errors, you may need to use more test vectors in the simulations.

The MAP Report (MRP) File

The MAP report (MRP) file is an ASCII (text) file containing information about the MAP command run. Although detailed information varies depending upon the device to which you have mapped, the format of the file is the same regardless of the device used.

Note The MRP file is formatted for viewing in a monospace (non-proportional) font. If the text editor you use for viewing the report uses a proportional font, the columns in the report do not line up correctly.

A sample MRP file is shown below. This is an abbreviated file—most MAP report files are considerably larger than the one shown below.

The report file is divided into a number of sections. Sections appear in the report file even if they are empty (that is, even if there are no messages that apply to them).

These are the sections in the MAP report file.

- Design Information—Shows your MAP command, line, the device to which the design has been mapped, and when the mapping was performed.
- Design Summary—Summarizes the mapper run, showing the number of errors and warnings, and how many of the resources in the target device are used by the mapped design.
- Table of Contents—Lists the remaining sections of the MAP report.
- Errors (Section 1)—Shows any errors generated as a result of the following.
  - Errors associated with the logical DRC tests performed at the beginning of the mapper run. These errors do not depend on the device to which you are mapping.
  - Errors the mapper discovers (for example, a pad is not connected to any logic, or a bidirectional pad is placed in the design but signals only pass in one direction through the pad). These errors may depend on the device to which you are mapping.
  - Errors associated with the physical DRC run on the mapped design.
- Warnings (Section 2)—Shows any warnings generated as a result of the following.
  - Warnings associated with the logical DRC tests performed at the beginning of the mapper run. These warnings do not depend on the device to which you are mapping.
  - Warnings the mapper discovers. These warnings may depend on the device to which you are mapping.
  - Warnings associated with the physical DRC run on the mapped design.
- Design Attributes (Section 3)—Shows any attributes (properties) specified when the design was created. Some of these attributes also appear as physical constraints in the physical constraints file (PCF) produced by the mapper run.
- Removed Logic Summary (Section 4)—Summarizes the number of blocks and signals removed from the design. The section reports on these kinds of removed logic.

- Blocks trimmed—A “trimmed” block is a block removed because it is along a path that has no driver or no load. Trimming is recursive; that is, if Block A becomes unnecessary because logic to which it is connected has been trimmed, then Block A is also trimmed.
- Blocks removed—A “removed” block is removed because it can be eliminated without changing the operation of the design. Removal is recursive; that is, if Block A becomes unnecessary because logic to which it is connected has been removed, then Block A is also removed.
- Blocks optimized—An “optimized” block is a block removed because its output remains constant regardless of the state of the inputs (for example, an AND gate with one input tied to ground). Logic generating an input to this optimized block (and to no other blocks) is also removed, and appears in this section.
- Signals removed—Signals that were removed because they were attached only to removed blocks.
- Signals merged—A signal is merged when two signals are combined because a component separating them was removed.

- Removed Logic (Section 5)—Describes in detail all logic (design components and nets) removed from the input NGD file when the design was mapped. The preceding description of Section 4 defines the types of logic removed. More generally, logic may be removed because
  - A design uses only part of the logic in a library macro.
  - The design has been mapped even though it is not yet complete.
  - The mapper has optimized the design logic.
  - Unused logic has been created in error during schematic entry.

  This section also indicates which nets were merged (that is, two nets were combined when a component separating them was removed).

In this section, if the removal of a signal or symbol results in the subsequent removal of an additional signal or symbol, the line describing the subsequent removal is indented. This indentation is repeated as a chain of related logic is removed. To quickly locate the cause for the removal of a chain of logic, look above the entry in which you are interested and locate the top-level line, which is not indented.

- Added Logic (Section 6)—Describes any logic that was added to the design by the mapper. For example, logic is added when a design contains global reset buffers but the device to which you are mapping does not have global reset buffers. The mapper adds the necessary logic to perform the global reset function
- Expanded Logic (Section 7)—If enabled, describes the mapping of logic that has been added to the database to resolve certain design blocks (for example, LogiBLOX modules).

  By default this section is empty, since the section may contain thousands of lines and the information is not needed by the majority of users. To create this section, select the -detail option
- Symbol Cross Reference (Section 9)—If enabled, shows where symbols in the logical design were mapped in the physical design.

  By default this section is empty, since the section may contain thousands of lines and the information is not needed by the majority of users. To create this section, select the -detail option.

- IOB Properties (Section 10)—Lists each IOB to which the user has supplied constraints along with the applicable constraints. The possible IOB properties are shown in the following table; the applicability of the properties and options varies from one architecture to another. The following table applies only to the XC4000X, Spartan/XL architectures.

Table 8-2 IOB Properties

Property	Meaning	Options
SLEW	Output slew rate	SLOW or FAST
PULLUP	Enable pull-up resistor	N/A
PULLDOWN	Enable pull-down resistor	N/A
FF/LATCH	Input flip-flop/latch data source	NODELAY, MEDDELAY, or SYNC
SYNC	Fast capture latch data source	NODELAY or MEDDELAY
DRIVE	Drive value on output pads	12 or 24 ma.

- RPMs (Section 11)—Indicates each RPM (Relationally Placed Macro) used in the design, and the number of device components used to implement the RPM.
- Guide Report (Section 12)—If you have mapped using a guide file, shows the guide mode used (EXACT or LEVERAGE) and the percentage of objects that were successfully guided.
- Area Group Summary (Section 13)— The mapper summarizes results for each area group. MAP uses area groups to specify a group of logical blocks that are packed into separate physical areas.

A sample MAP Report (MRP) file is shown below.

Note The MAP Report is formatted for viewing in a monospace (non-proportional) font. If the text editor you use for viewing the report uses a proportional font, the columns in the report do not line up correctly.

```
Xilinx Mapping Report File for Design 'area_con14'
Copyright (c) 1995-2000 Xilinx, Inc.  All rights reserved.
```

```
Design Information
------------------
Command Line   : map -u area_con14.ngd
Target Device  : xv50
Target Package : bg256
Target Speed   : -6
Mapper Version : virtex -- HEAD
Mapped Date    : Tue Mar  7 13:30:28 2000

Design Summary
--------------
  Number of errors:           0
  Number of warnings:         56
  Number of Slices:           0 out of  768   0%
  Number of Slices containing
    unrelated logic:            0 out of  0     0%
  Number of 4 input LUTs:     0 out of 1,536  0%
  Number of Block RAMs:       2 out of 8      25%
Total equivalent gate count for design:  32,768

Table of Contents
-----------------
Section 1 - Errors
Section 2 - Warnings
Section 3 - Design Attributes
Section 4 - Removed Logic Summary
Section 5 - Removed Logic
Section 6 - Added Logic
Section 7 - Expanded Logic
Section 8 - Signal Cross-Reference
Section 9 - Symbol Cross-Reference
Section 10 - IOB Properties
Section 11 - RPMs
Section 12 - Guide Report
Section 13 - Area Group Summary

Section 1 - Errors
------------------

Section 2 - Warnings
--------------------
```

```
WARNING:DesignRules:368 - Netcheck: Sourceless. Net clk has no
source.
WARNING:DesignRules:368 - Netcheck: Sourceless. Net en has no
source.
WARNING:DesignRules:368 - Netcheck: Sourceless. Net rst has no
source.
WARNING:DesignRules:368 - Netcheck: Sourceless. Net we has no
source.
WARNING:DesignRules:368 - Netcheck: Sourceless. Net addr11 has no
source.
.
.
.
WARNING:DesignRules:367 - Netcheck: Loadless. Net m2_m2_doa0 has no
load.
WARNING:DesignRules:367 - Netcheck: Loadless. Net m2_m2_doa1 has no
load.
WARNING:DesignRules:367 - Netcheck: Loadless. Net m2_m2_dob0 has no
load.
WARNING:DesignRules:367 - Netcheck: Loadless. Net m2_m2_dob1 has no
load.

Section 3 - Design Attributes
-----------------------------

Section 4 - Removed Logic Summary
---------------------------------

Section 5 - Removed Logic
-------------------------

Section 6 - Added Logic
-----------------------

Section 7 - Expanded Logic
--------------------------
To enable this section, set the detailed map report option and rerun
map.

Section 8 - Signal Cross-Reference
----------------------------------
```

```
To enable this section, set the detailed map report option and rerun
map.

Section 9 - Symbol Cross-Reference
----------------------------------
To enable this section, set the detailed map report option and rerun
map.

Section 10 - IOB Properties
---------------------------

Section 11 - RPMs
-----------------

Section 12 - Guide Report
-------------------------
Guide not run on this design.

Section 13 - Area Group Summary
-------------------------------
  AREA_GROUP AG_ONE
    RANGE: RAMB4_R1C1:RAMB4_R3C1
  No COMPRESSION specified for AREA_GROUP AG_ONE
    Number of BlockRAMs:       2 out of     3 66%
```

Halting MAP

To halt MAP, enter CONTROL-C (on a workstation) or CONTROL-BREAK (on a PC). On a workstation, make sure that when you enter CONTROL-C the active window is the window from which you invoked the mapper. The operation in progress is halted. Some files may be left when the mapper is halted (for example, a MAP report file or a physical constraints file), but these files may be discarded since they represent an incomplete operation.

Chapter 9

LCA2NCD

LCA2NCD is compatible with the following families.

- XC4000E
- XC3000A / L
- XC3100A / L
- XC5200

This chapter describes LCA2NCD. The chapter contains the following sections.

- "LCA2NCD"
- "LCA2NCD Syntax"
- "LCA2NCD Files"
- "LCA2NCD Options"
- "Translating Unnamed Components"

LCA2NCD

Earlier releases of the Xilinx Development System stored the physical design in a Logic Cell™ Array (LCA™) file. The current Xilinx Development System tools operate on physical designs in the NCD (Circuit Description) format. LCA files are ASCII (human-readable) files; NCD files are binary (machine-readable) files.

LCA2NCD converts an LCA file to an NCD file, as shown in the following figure. The NCD file produced by LCA2NCD can be placed and routed, viewed in the FPGA Editor, analyzed for timing, and back-annotated in the current Xilinx Development System.

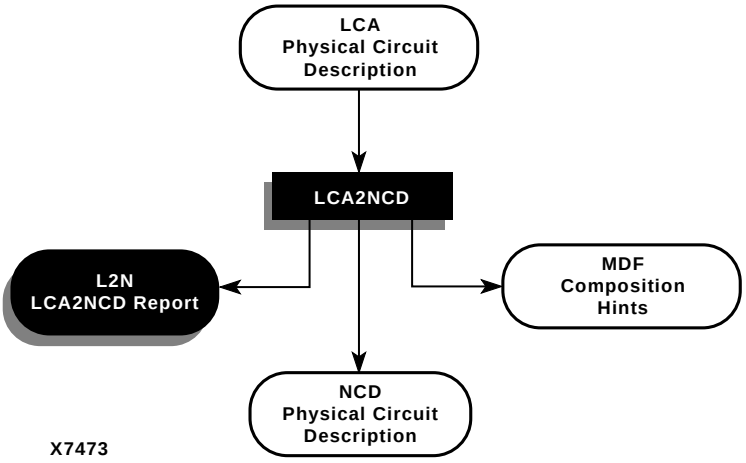


Figure 9-1 LCA2NCD File Conversion

LCA2NCD Syntax

The following syntax converts an LCA file to an NCD file.

lca2ncd [*options*] *lca_file*[**.lca**] [*ncd_file*][**.ncd**]]

options can be any number of the options listed in the "LCA2NCD Options" section. They do not need to be listed in any particular order. Separate multiple options with spaces.

lca_file is the LCA file to be converted. If you enter a file name with no extension, LCA2NCD looks for a file with an .lca extension and the name you specified.

ncd_file is an optional name for the output NCD file. The output file name, its extension, and its location are determined in the following way.

- If you do not specify an output file name, the output file has the same name as the input file, with an .ncd extension.
- If you specify an output file name with no extension, LCA2NCD appends the .ncd extension to the file name.
- If you specify a file name with an extension other than .ncd, you get an error message and LCA2NCD does not run.

- If you do not specify a full path name, the output file is placed in the directory from which you ran LCA2NCD.

LCA2NCD Files

The input files that LCA2NCD requires and the output files that LCA2NCD generates are described below.

Input Files

Input to LCA2NCD consists of a Xilinx LCA file. This is a mapped design file generated in a previous revision of the Xilinx Development System. The file may also contain placement and routing information

Output Files

Output from LCA2NCD consists of the following files.

- NCD file—This file is a physical description of the design in terms of Xilinx components, such as logic cells and I/O cells.
- MDF file—The MAP Directive File describes how logic was decomposed when the design was originally mapped. The MDF file is used for guided mapping with the current Xilinx Development System software. In guided mapping, the file enables MAP to recreate the decompositions chosen when the design was first mapped. This file is only created if there are Mapper directives in the LCA file.
- L2N file—This is a report file containing information about the LCA2NCD run.

LCA2NCD Options

Following is a description of the command line options and how they affect the behavior of LCA2NCD.

–p (Placement Only)

If you specify the –p option, LCA2NCD includes placement information from the input LCA file in the output NCD file, but no routing information.

–f (Execute Commands File)

-f *command_file*

The –f option executes the command line arguments in the specified *command_file*. For more information on the –f option, see the "–f Option" section of the "Introduction" chapter.

–w (Overwrite Existing File)

The –w option overwrites the output NCD file if it already exists. By default (no –w specified), LCA2NCD does not overwrite an existing file

Translating Unnamed Components

A Xilinx LCA file can contain unnamed components. Components that are unnamed in an LCA file are dynamically named when used in the Xilinx Development System tools; that is, component names change depending on whether the components are placed or unplaced.

In an LCA file, components are assigned names using the NmBlk construct. Although a component in an LCA file does not have to be assigned a name, both the FPGA Editor and PAR require something by which to refer to the component. In the Xilinx Development System tools, the name applied to a component is dynamic—the same component has a different name when it is placed or unplaced.

If an unnamed component is placed, it is referred to in the following way.

$*sitename_id*

sitename is the site in which the component is placed. *id* is an integer.

If an unnamed component is unplaced, it is referred to in the following way.

$*typename_id*

typename is the type of component, such as CLB, IOB, or TBUF. *id* is an integer.

If an unplaced component is placed in PAR or the FPGA Editor, or if a placed component is unplaced, the string by which it is referred to changes.

The FPGA Editor allows you to rename components. If you use the FPGA Editor to assign a name to an unnamed component, the name you supply then remains with the component, whether the component is placed or unplaced.

Chapter 10

Physical Constraints (PCF) File

This chapter describes the Physical Constraints File (PCF). The chapter contains the following sections

- "The PCF File"
- "Interaction Between Constraints"

The PCF File

The NGD file produced when a design netlist is read into the Xilinx Development System may contain a number of logical constraints. These constraints originate in any of these sources.

- An attribute assigned within a schematic or HDL file.
- A constraint entered in a UCF (User Constraints File).
- A constraint appearing in an NCF (Netlist Constraints File) produced by a CAE vendor toolset.

Logical constraints in the NGD file are read by MAP. MAP uses some of the constraints to map the design, and converts other logical constraints to physical constraints. MAP then writes these physical constraints into a Physical Constraints File (PCF).

The PCF file is an ASCII file containing two separate sections: a section for those physical constraints created by the mapper and a section for physical constraints entered by the user. The mapper section is rewritten every time you run the mapper. Mapper-generated physical constraints appear first in the file, followed by user physical constraints. This order dictates that in the event of conflicts between mapper-generated and user constraints, user constraints are last-read and override. The mapper-generated section of the file is preceded by a **`SCHEMATIC START`** notation on a separate line.

The end of this section is indicated by **SCHEMATIC END**, also on a separate line. User-generated constraints, such as timing constraints, should always be entered after **SCHEMATIC END**.

You can write user constraints directly into the file or you can write them indirectly (or undo them) from within the FPGA Editor. (For more information on constraints in the FPGA Editor, see the *FPGA Editor Reference/User Guide*).

The PCF file is an optional input to PAR, the FPGA Editor, TRACE, NGDAnno, and BitGen (see the following figure).

The file may contain any number of constraints and any number of comments in any order. A comment consists of either a pound sign (#) or double slashes (/ /) followed by any number of other characters up to a new line. Illegal constraints are automatically commented out by the program.

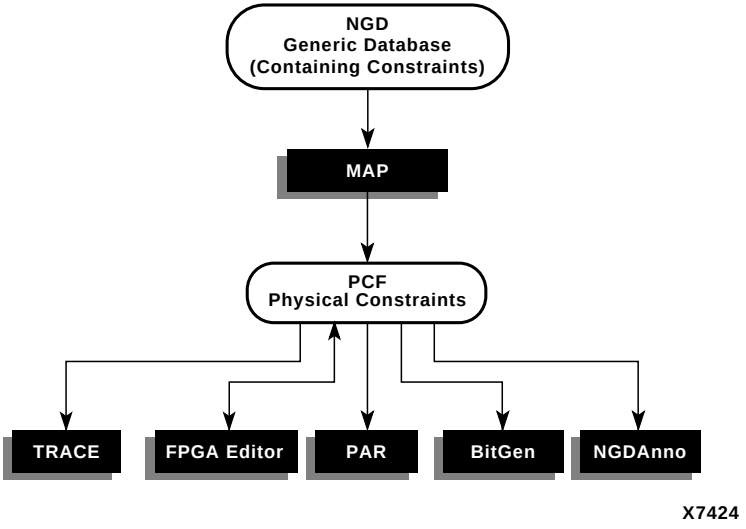


Figure 10-1 PCF File Flow

Interaction Between Constraints

Schematic constraints are placed at the beginning of the PCF file by MAP. The start and end of this section is indicated with **SCHEMATIC START** and **SCHEMATIC END**, respectively. Because of a "last-read" order, all constraints that you enter in this file should come after **SCHEMATIC END**.

Note You are not prohibited from entering a user constraint before the schematic constraints section, but if you do, a conflicting constraint in the schematic-based section may override your entry.

Every time a design is remapped, the schematic section of the PCF file is overwritten by the mapper. The user constraints section is left intact, but certain constraints may be invalid because of the new mapping.

Chapter 11

DRC—Physical Design Rule Check

This program is compatible with the following families.

- Spartan / XL / -II
- Virtex / -E / -II
- XC4000E / L / EX / XL / XV / XLA
- XC3000A / L
- XC3100A / L
- XC5200

The chapter contains the following sections.

DRC

The physical Design Rule Check (DRC) consists of a series of tests to discover physical errors and some logic errors in the design. Three Xilinx Development System modules employ physical DRC. The physical DRC is used in the following ways.

- MAP automatically runs physical DRC after it has mapped the design.
- PAR (Place and Route) automatically runs physical DRC on nets when it routes the design.
- You can run physical DRC from within the FPGA Editor. The DRC also runs automatically after certain FPGA Editor operations (for example, when you edit a logic cell or when you manually route a net). For a description of how the DRC works within the FPGA Editor, see the "Verifying Your Design" section in the *FPGA Editor Guide*.
- BitGen, which creates a a BIT file for programming the device, automatically runs physical DRC.
- You can run physical DRC from the UNIX or DOS command line.

DRC Syntax

To run DRC, enter the following on the UNIX or DOS command line.

```
drc [options] file_name
```

options can be any number of the DRC options listed in the "DRC Options" section. They do not need to be listed in any particular order. Separate multiple options with spaces.

file_name is the name of the NCD file on which DRC is to be run.

DRC Files

This section describes the DRC input and output files.

Input File

The input to DRC is an NCD file. The NCD file is a mapped, physical description of your design.

Output File

The output of DRC is a TDR file. The TDR file is an ASCII DRC report. The contents of this file is determined by the options you select for the DRC command.

DRC Options

This section describes the options that are available for the DRC command line.

–e (Error Report)

The –e option produces a report containing details about errors only. No details are given about warnings.

–o (Output file)

-o *outfile_name*

The –o option overrides the default output report file *file_name*.tdr with *outfile_name*.tdr.

–s (Summary Report)

The –s option produces a summary report only. The report indicates the number of errors and warnings found but does not supply any details about them.

–v (Verbose Report)

The –v option reports all warnings and errors. This is the default option for DRC.

–z (Report Incomplete Programming)

The –z option reports incomplete programming as errors. Certain DRC violations are considered errors when the DRC runs as part of the BitGen command but are considered warnings at all other times the DRC runs. These violations usually indicate the design is incompletely programmed (for example, a logic cell has been only partially programmed or a signal has no driver). The violations would create errors if you tried to program the device, so they are reported as errors when BitGen creates a BIT file for device programming. If you run DRC from the command line without the –z option, these violations are reported as warnings only. With the –z option, these violations are reported as errors.

DRC Types

Physical DRC can perform four types of checks.

- Net check—This check examines one or more routed or unrouted signals and reports any problems with pin counts, tristate buffer inconsistencies, floating segments, antennae, and partial routes.
- Block check—This check examines one or more placed or unplaced components and reports any problems with logic, physical pin connections, or programming.
- Chip check—This check examines a special class of checks for signals, components, or both at the chip level, such as placement rules with respect to one side of the device.
- All checks—This check performs net, block, and chip checks.

When you run DRC from the command line (as described in the previous sections), it automatically performs net, block, and chip checks.

In the FPGA Editor, you can run the net check on selected objects or on all of the signals in the design. Similarly, the block check can be performed on selected components or on all of the design's components. When you check all components in the design, the block check performs extra tests on the design as a whole (for example, tristate buffers sharing long lines and oscillator circuitry configured correctly) in addition to checking the individual components. In the FPGA Editor, you can run the net check and block check separately or together.

For a description of how the DRC works within the FPGA Editor, see the "Verifying Your Design" section of the *FPGA Editor Guide*.

DRC Errors and Warnings

A DRC error indicates a condition in which the routing or component logic will not operate correctly (for example, a net without a driver or a logic block that is incorrectly programmed). A DRC warning indicates a condition where the routing or logic is incomplete (for example, a net is not fully routed or a logic block has been programmed to process a signal but there is no signal on the appropriate logic block pin).

Certain messages may appear as either warnings or errors, depending on the application and signal connections. For example, in a net check, a pull-up not used on a signal connected to a decoder generates an error message. A pull-up not used on a signal connected to a tristate buffer only generates a warning.

Incomplete programing (for example, a signal without a driver or a partially programmed logic cell) is reported as an error when the DRC runs as part of the BitGen command, but is reported as a warning when the DRC runs as part of any other program. The –z option to the DRC command reports incomplete programming as an error instead of a warning. For a description of the –z DRC option, see the "–z (Report Incomplete Programming)" section.

Chapter 12

PAR—Place and Route

This program is compatible with the following families.

- XC3000A / L
- XC3100A / L
- XC4000E / L
- XC4000EX / XL / XV / XLA
- XC5200
- Spartan / XL / -II
- Virtex / -E / -II

The chapter contains the following sections.

- "Halting PAR"

PAR

After a design has undergone the necessary translation to bring it into the NCD (Circuit Description) format, it is ready for placement and routing. This phase is done by PAR (Xilinx's Place and Route program). PAR takes an NCD file, places and routes the design, and outputs an NCD file which is used by the bitstream generator (BitGen). The output NCD file can also act as a guide file when you reiterate placement and routing for a design to which minor changes have been made after the previous iteration.

In the Xilinx Development System, PAR places and routes a design using a combination of two methods.

- Cost-based—This means that placement and routing are performed using various cost tables which assign weighted values to relevant factors such as constraints, length of connection, and available routing resources. Cost-based placement and cost-based routing are further described in the "PAR Operation" section.
- Timing-Driven—The Xilinx timing analysis software enables PAR to place and route a design based upon your timing constraints. Timing-driven PAR is described in the "PAR and the Timing Analysis Software" section.

Flow through the PAR module is shown in the following figure. The figure shows a PAR run that produces a single output design file.

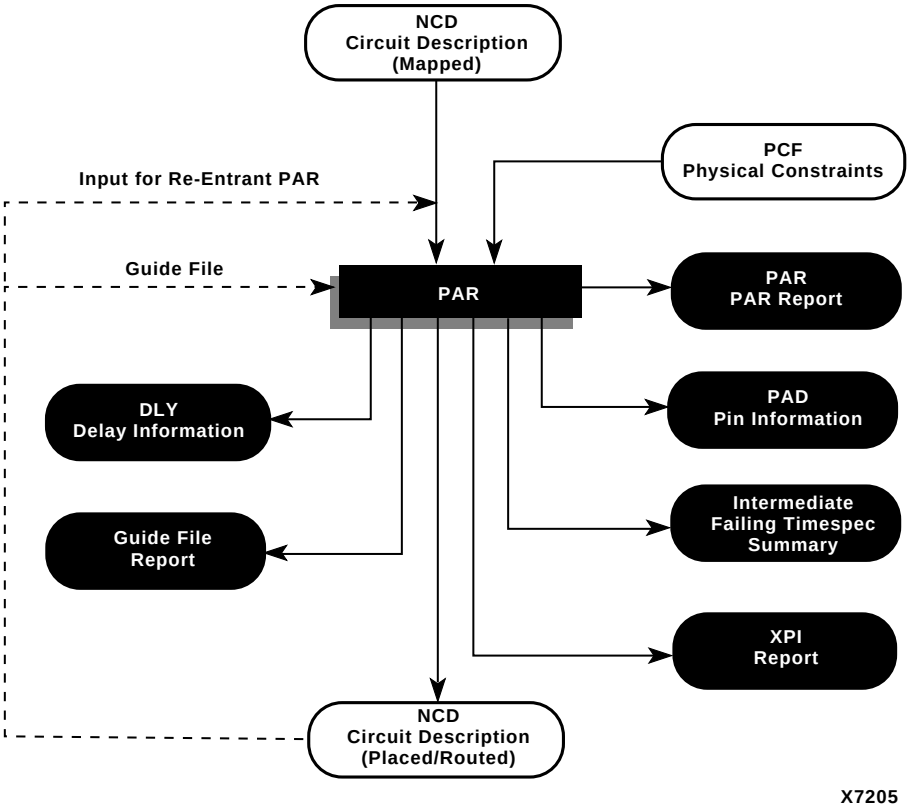


Figure 12-1 PAR Flow

PAR and the Timing Analysis Software

Timing-driven PAR is based upon Xilinx's timing analysis software, an integrated static timing analysis tool (that is, it does not depend on input stimulus to the circuit). This means that placement and routing are executed according to timing constraints that you specify in the beginning of the design process. The timing analysis software interacts with PAR to ensure that the timing constraints you impose on the design are met.

To use timing-driven PAR, you can specify your timing constraints in any of these ways.

- You can enter the timing constraints as properties in a schematic capture or HDL design entry program.
- You can write your timing constraints into a User Constraints File (UCF). This file is processed by NGDBuild when the logical design database is generated.

  To avoid manually entering timing constraints in a UCF file, use the Xilinx Constraints Editor, a tool that greatly simplifies constraint creation. For a detailed description of how to use the editor, see the Constraints Editor Guide.
- You can enter the timing constraints in the Physical Constraints File (PCF), a file that is generated by MAP. The PCF file contains any timing constraints specified using the two previously described methods and any additional constraints you enter directly in the file.

Timing-driven placement and timing-driven routing are automatically invoked if PAR finds timing constraints in the physical constraints file. The physical constraints file serves as input to the timing analysis software. The timing constraints supported by the Xilinx Development System are described in the "Using Timing Constraints" chapter.

Note Depending upon the types of timing constraints specified and the values assigned to the constraints, PAR run time may be increased.

When PAR is complete, you can verify that the design's timing characteristics (relative to the physical constraints file) have been met by running TRACE (Timing Reporter and Circuit Evaluator), Xilinx's timing verification and reporting utility. TRACE, which is described in detail in the "TRACE" chapter, issues an ASCII report showing any timing warnings and errors and other information relevant to the design. There is a terse summary report of timing in the PAR report also.

Note If you are going to run a design without timing constraints, better circuit performance most likely can be obtained by enabling the Delay Based Cleanup router pass. Alternatively, consider running timing driven PAR by supplying timing constraints with the input design.

Automatic Timespecing

For Spartan-II and Virtex/-E designs, PAR performs automatic timespecing if it detects no timing constraints. This feature is invoked only if you set the effort levels at 3, 4 or 5. Effort level 2 is the default at which PAR runs. For more information on effort levels, see the "–l (Overall Effort Level)" section. Automatic timespecing can be suppressed. See the "–x (Ignore Timing Constraints)" section for details.

When automatic timespecing is invoked, PAR analyzes the design for clock nets and attempts to increase the frequency of clocks. If you were to increase the clock frequency using a timing-driven flow, you would perform multiple runs of PAR with progressively higher frequencies. Each run would require you to increase the frequency specifications on clocks in your design. PAR would attempt to meet these specifications, not improve on them. You might continue to increase the frequency specifications until PAR can no longer meet them. At this point, your design would achieve optimal clock frequency. In contrast, automatic timespecing allows you to achieve good clock frequency results in the shortest possible time.

Specifying timing constraints in your design file can still yield the best possible runtime and frequency. However, automatic timespecing provides significantly improved runtime and clock frequency compared with non timing driven mode.

PAR Syntax

The following syntax places and routes your design.

par [*options*] *infile*[**.ncd**] *outfile pcf_file*[**.pcf**]

Options can be any number of the PAR options listed in the "PAR Options" section. They do not need to be listed in any particular order. Separate multiple options with spaces.

Infile is the design file you wish to place and/or route. The file must have an .ncd extension, but you do not have to specify the .ncd extension on the command line.

Outfile is the target design file which is written after PAR is finished. If the command options you specify yield a single output design file, *outfile* has an extension of .ncd or .dir. An .ncd extension generates an output file in NCD format; the .dir extension directs PAR to create a

directory in which to place the output file (in NCD format). If the command options you specify yield more than one output design file (that is, you enter the –n option described in the "PAR Options" section), *outfile* must have an extension of .dir. The multiple output files (in NCD format) are placed in the directory with the .dir extension.

If the file or directory you specify already exists, you get an error message and the operation does not run. You can override this protection and automatically overwrite existing files by using the –w option (described in the "PAR Options" section).

Pcf_file is a physical constraints file. The file contains the constraints you entered during design entry, constraints you added using the UCF (User Constraints File), and constraints you added directly in the PCF file. If you do not enter the name of a physical constraints file on the command line and the current directory contains an existing physical constraints file with the *infile* name and a .pcf extension, PAR uses the PCF file.

PAR Files

This section describes the PAR input and output files.

Input Files

Input to PAR consists of the following files.

- NCD file—a mapped design.
- PCF file—an ASCII file containing constraints based on attributes in the schematic or on constraints you have placed in a UCF file. A complete list of constraints can be found in the "Attributes, Constraints, and Carry Logic" chapter of the *Libraries Guide*. PAR supports all of the timing constraints described in that chapter.
- Guide NCD file—an optional template file for placing and routing the design.

Output Files

Output from PAR consists of the following files.

- NCD file—a placed and routed design file (may contain placement and routing information in varying degrees of completion).

- PAR file—a PAR report including summary information of all placement and routing iterations.
- DLY file—a file containing delay information for each net in the design.
- PAD file—a file containing I/O pin assignments.
- Guide Report File—a file created if you use the -gf option
- Intermediate Failing Timespec Summary— a summary generated for failing timing specifications
- Xilinx Par Information— This XPI report displays whether or not the design routed and if timing specifications were met.

PAR Options

You can customize the PAR operation by specifying options when you run PAR. You can have PAR place without routing. You can have PAR perform a single placement, or perform a number of placements using different cost tables. You can specify an effort level to indicate to PAR whether the design is simple or complex. You can also specify the maximum number of passes the router may perform and the number and type of cleanup passes the router runs.

PAR options are entered on the command line in any order, preceded by a hyphen (–), and separated by spaces. You must enter options in lower case letters. For those options that require an additional parameter, the option and the parameter must be separated by spaces or tabs. Options that do not require an additional parameter may be grouped together preceded by a single hyphen (for example, –rwx is the same as –r –w –x).

Following is a description of the command line options and how they affect the behavior of PAR. If you run PAR with illegal options or do not specify an input file, a brief listing of the supported options and their functions is printed on the screen. If you want to view all of the options, type the following on the command line.

```
par | more
```

This allows you to scroll through the options.

OR

```
par > filename
```

This redirects the options to a file that you specify.

–c (Number of Cost-Based Router Cleanup Passes)

-c *cost_passes*

If you run both cost-based cleanup passes and delay-based cleanup passes (see –d and –e options below), the cost-based passes run before the delay-based passes. The valid range of *cost_passes* is 0–5. The default setting for –c is 1 for all devices except Spartan-II and Virtex/-E/-II, for which the default is 0.

Following are some strategies for using the cleanup routers (either cost or delay based).

- On non-timing driven runs, cleanup routing can significantly improve delays and is, in fact, mandatory for XC3000 devices.
- If cost-based cleanup does not yield the desired performance on a non-timing driven run, running a delay-based cleanup pass may often significantly improve circuit performance.
- For timing-driven runs, the cleanup passes can improve timing on those elements of the design that are not covered by timing constraints.
- Also, for designs in which normal iterative routing is not quite making the timing goal (but is somewhat close, say 3 - 5%) a delay-based cleanup pass can sometimes reorganize the routing enough such that follow-up re-entrant iterative routing passes are then able to meet timing.

Note The -c option is not recommended for use with Virtex/-E/-II or Spartan-II. The evaluation of this option with these architectures indicates that the option creates much longer runtimes with hardly any improvement.

–d (Number of Delay-Based Router Cleanup Passes)

-d *delay_passes*

If you do not use the –d option, the router does not run any delay-based cleanup passes (described in the “Routing” section). If you run both delay-based cleanup passes and cost-based cleanup passes (see –c option above), the cost-based passes run before the delay-based passes. Typically, the first delay-based cleanup pass produces the

greatest improvement, with less improvement on each successive pass. It is also possible that delay passes do not show any improvement.

The valid range of *delay_passes* is 0–5, and the default is 0.

If you want to run delay-based cleanup passes only on designs that have been routed to completion (100% routed), use the –e option (described below) instead of the –d option.

Note The -d option is not recommended for use with Virtex/-E/II or Spartan-II. The evaluation of this option with these architectures indicates that the option creates much longer runtimes with little improvement.

-dfs (Thorough timing analysis of paths)

The -dfs option specifies that PAR utilize depth-first search timing analysis, which analyzes all paths covered by timing constraints in order to perform timing-driven place and route. This method is more thorough than the default method (-kpaths) and may result in longer PAR runtimes. See the "Output Files" section for a discussion of the connection-based method.

–e (Delay-based cleanup passes—Completely Routed Designs)

-e *number*

The –e option operates in the same way as the –d option described previously, but the –d option runs on *all* output designs produced by the PAR run, while the –e option only runs on those output designs which have been routed to completion. The *number* of passes is 0–5, and the default is 0.

Note This option is not recommended for use with Virtex/-E/-II or Spartan-II. The evaluation of this option with these architectures indicates that the option creates much longer runtimes with little improvement.

–f (Execute Commands File)

-f *command_file*

The –f option executes the command line arguments in the specified *command_file*. For more information on the –f option, see the "–f Option" section of the "Introduction" chapter.

–gf (Guide NCD File)

-gf *guide_file*

The –gf option specifies the name of an NCD file (from a previous MAP or PAR run) to be used as a guide for this PAR run. The guide file is an NCD file which is used as a template for placing and routing the input design. For more information on the guide file, see the "Guided PAR" section.

–gm (Guide Mode)

-gm {**exact** | **leverage**}

The –gm option specifies the form of guided placement and routing PAR uses—exact or leveraged. The default is exact mode. For more information on the guide modes, see the "Guided PAR" section.

You specify the NCD to use as a guide file by entering a –gf option (see the "Output Files" section) on the PAR command line. If you do not specify a guide file, PAR is guided by the placement and routing information in the input NCD file. The "Guided PAR" section describes how PAR operates if no guide file is specified.

–i (Number of Router Iterations)

-i *route_passes*

Run a maximum number of passes of the router, stopping earlier only if the routing goes to 100% completion and all constraints are met. Each pass is a single attempt to route a placement to completion, and the screen displays a message for each pass.

The valid range of *route_passes* is 0–2000. If you do not use the –i option, the router runs until it either routes to 100% completion and meets its timing constraints or intelligently determines it will not succeed.

-k (Re-Entrant Routing)

The –k option runs re-entrant (also called incremental) routing. Routing begins with the existing placement and routing left in place. Re-entrant routing is useful if you wish to manually route parts of the design and then continue automatic routing, if you halted the route prematurely (for example, with a Control-C) and wish to resume, or if you wish to run additional route or delay reduction passes.

-kpaths (Faster Analysis of Paths)

The non-enumerative connection-based method (the default method) has a runtime proportional to the size of the design, unlike the DFS method, which has a runtime proportional to the number of paths in the design.

There are two significant differences between the connection-based method and the DFS method.

- The DFS method analyzes all paths except those that actually contain a circuit cycle, including paths that contain connections that cause a circuit cycle for other paths in the circuit. The connection-based method may not analyze these paths depending on circuit topology. Consider the following example circuit.

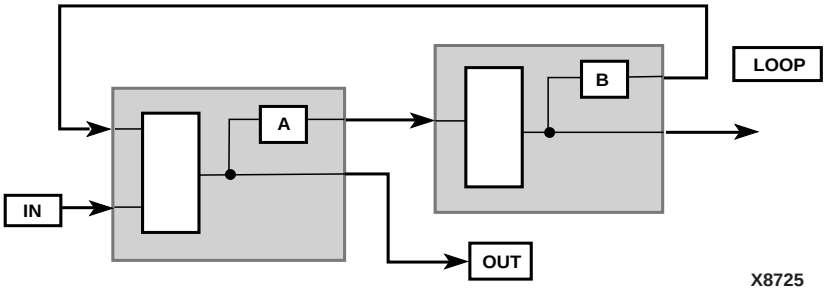


Figure 12-2 Circuit Cycles

The DFS method traces the path from IN, through A, through the signal LOOP, back to the left-most logic block and to the signal OUT. The new connection-based method may not trace this path because a combinatorial cycle exists at the output of A.

- The DFS method removes false paths from a design that requires contending tristate enable signals. The connection-based method does not perform this optimization which means that it may analyze some paths that are statically false based on tristate enable signals. Consider the following circuit.

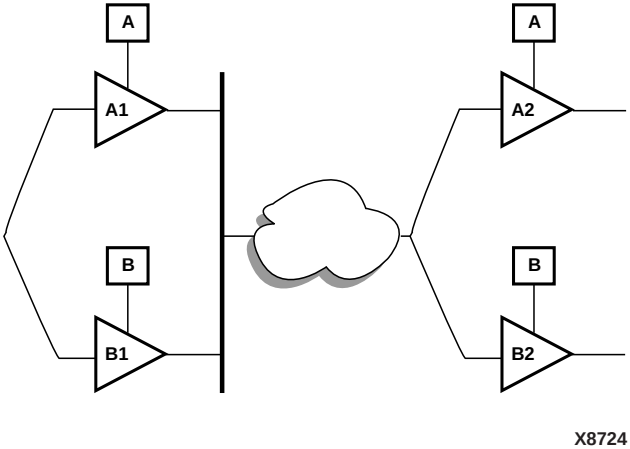


Figure 12-3 Tristate Buffer Paths

A signal can pass through four paths in the preceding circuit but two of the paths are false (A1 to B2 and B1 to A2). In order for a signal to pass through the upper left tristate buffer A1, the enable signal A must be true. In order to prevent a bus contention on the A1 output, the enable signal B must be false. Since buffer B2 is also controlled by the enable signal B, the path through A1 cannot pass through B2 (because when A is enabled, B is disabled). The converse is also true, if B is enabled, the only valid path is from B1 to B2.

In the example circuit, the DFS method only considers true paths. The connection-based will trace the false paths and the true paths.

–l (Overall Effort Level)

`-l` *effort_level*

The –l option is identical to the –ol option. See the “Output Files” section.

–m (Multi-Tasking Mode)

-m *nodefile_name*

The –m option allows you to assign PAR multi-run jobs (specified with the –n option) to a group of nodes. See the "Turns Engine (PAR Multi-Tasking Option)" section.

–n (Number of PAR Iterations)

-n *iterations*

The –n option determines the number of iterations (place and route passes) performed at the effort level specified by the –l option.

Each iteration uses a different cost table when the design is placed and produces a different NCD file. If you enter -n 0, the software continues to place and route, stopping either after the design is fully routed or after completing the iteration at cost table 100 and meeting all timing constraints. If you don't specify the –n option, one place and route iteration runs.

If you specify a –t option, the iterations begin at the cost table specified by –t.

The valid range of *iterations* is 0–100, and the default is 1.

–ol (Overall Effort Level)

-ol *effort_level*

The –ol option sets the overall PAR effort level. The effort level specifies the level of effort PAR uses to place and route your design to completion and achieve your timing constraints.

There are five values for *effort_level*. Level 1 is the lowest level, and corresponds to the least complex design. Level 5 would be used on the most complex design. The level is not an absolute; it shows instead relative effort. After you use PAR for a while, you will be better able to estimate whether a design is simple or complex.

If you place and route a simple design at a complex level, the design is placed and routed properly, but the process takes more time than placing and routing at a simpler level. If you place and route a complex design at a simple level, the design may not route to completion or may route less completely (or with worse delay characteristics) than at a more complex level.

The *effort_level* range is 1–5, and the default level is 2.

The –ol level sets an effort level for placement and another effort level for routing. These levels also have a range of 1–5. The placement and routing levels set at a given –ol level depend on the device family in the NCD file. You can determine the default placer and router effort levels for a device family by reading the PAR Report file produced by your PAR run.

You can override the placer level set by the –ol option by entering a –pl (Placer Effort Level) option, and you can override the router level by entering a –rl (Router Effort Level) option.

Note On Spartan-II and Virtex/-E devices, automatic timespecing is performed if PAR does not detect timing constraints and the effort level is set at 3, 4, or 5. See the"Automatic Timespecing" section in this chapter for more information.

–p (No placement)

The –p option bypasses both constructive and optimizing placement (described in the "Placement" section). When you use this option, existing routes are ripped up before routing begins. You can, however, leave the existing routing in place if you use the –k option instead of the –p option.

–pl (Placer Effort Level)

-pl *placer_effort_level*

The –pl option sets the placer effort level. The effort level specifies the level of effort used when placing the design. Level 1 is the lowest level, and corresponds to the least complex design. Level 5 would be used on the most complex design. For a description of effort level, see the "Output Files" section.

The *placer_effort_level* range is 1–5, and the default level set if you do not enter a –pl option is determined by the setting of the –ol option.

This default varies depending on the device family in the input NCD file. You can determine the default placer effort level for a given –ol level and device family by reading the PAR Report file produced by your PAR run.

–r (No Routing)

Do not route the design.

–rl (Router Effort Level)

-rl *router_effort_level*

The –rl option sets the router effort level. The effort level specifies the level of effort used when routing the design. Level 1 is the lowest level and corresponds to the least complex design. Level 5 would be used on the most complex design. For a description of effort level, see the "Output Files" section.

The *router_effort_level* range is 1–5, and the default level set if you do not enter a –rl option is determined by the setting of the –ol option. This default varies depending on the device family in the input NCD file. You can determine the default router effort level for a given –ol level and device family by reading the PAR Report file produced by your PAR run.

–s (Number of Results to Save)

-s *number_to_save*

The –s option saves only the number of results you specify. If you do not use the –s option, all results are saved.

The –s option does not care how many iterations you performed or how many effort levels were used. It compares every result to every other result and leaves you with the best number of NCD files. The best outputs are determined by a score assigned to each output design. This score takes into account such factors as the number of unrouted nets, the delays on nets and conformance to your timing constraints. The lower the score, the better the design. This score is described in the "Scoring the Routed Design" section.

The valid range for *number_to_save* is 0–100, and the default –s setting (no –s option specified) saves all results.

–t (Starting Placer Cost Table)

-t *placer_cost_table*

The –t option specifies the cost table at which the placer starts (placer cost tables are described in the "Placement" section). If you do not specify the –t option, the PAR software starts at placer cost table 1. If cost table 100 is reached, placement does not begin at 1 again, even if command options specify that more placements should be performed. Cost tables are not an ordered set. There is no correlation between a cost table's number and its relative value.

The *placer_cost_table range* is 1–100, and the default is 1.

–ub (Use Bonded I/Os)

If you do not specify the –ub option, I/O logic that MAP has identified as internal can only be placed in unbonded I/O sites.

If the –ub option is specified, PAR can place this internal I/O logic into bonded I/O sites in which the I/O pad is not used. The option also allows PAR to route through bonded I/O sites.

If you use the –ub option, you must make sure this logic is not placed in bonded sites connected to external signals, power, or ground. You can prevent this condition by placing PROHIBIT constraints on the appropriate bonded I/O sites.

–w (Overwrite Existing Files)

If the specified output file already exists, overwrite the existing file (including the input file).

If the specified output is a directory, overwrite files in the directory. With this option, any PAR, PAD, and DLY files generated overwrite existing PAR, PAD, and DLY files.

–x (Ignore Timing Constraints)

If you do not specify the –x option, the PAR software automatically runs a timing-driven PAR run if any timing constraints are found in the physical constraints file. If you do specify –x, timing-driven PAR is not invoked in any case.

The –x option might be used if you have timing constraints specified in your physical constraints file, but you want to execute a quick PAR run without using the timing-driven PAR feature, to give you a rough idea of how difficult the design is to place and route.

Note On Spartan-II and Virtex/-E devices, using the -x option suppresses the automatic timespecing feature. For more information, see the "Automatic Timespecing" sectionin this chapter.

Options, default values, and ranges are summarized below.

Option	Default Setting	Range
–c *number*	The default setting for –c is 1 for all devices except Spartan II, Virtex/-E/-II, for which the default is 0.	0–5
–d *number*	0 (No delay-based router cleanup passes)	0–5
–dfs	Run connection- based method (No –dfs, -kpaths is the default)	N/A
–e *number*	0 (No delay-based router cleanup passes on completely routed designs)	0–5
–gf	No guide file	N/A
–gm [**leverage** | **exact**]	Exact	N/A
–i *number*	Run until completion or until router decides it can not complete	0–2000
–k	Run placement (Do not run re-entrant routing)	N/A
-kpaths	Run connection-based method (-kpaths is the default)	N/A
–l *number*	2 (Overall effort level 2)	1–5
–m *nodefile_name*	Do not run the Turns Engine	N/A
–n *number*	1 (One place and route iteration)	0–100
–ol *number*	2 (Overall effort level 2)	1–5
–p	Run placement	N/A
–pl *number*	Determined by –ol setting	1–5
–r	Run router	N/A
–rl *number*	Determined by –ol setting	1–5

Option	Default Setting	Range
–s *number*	Save all	1–100
–t *number*	1 (Start placer at cost table 1)	1–100
–ub	Do not use bonded I/Os	N/A
–w	Do not overwrite	N/A
–x	Use timing constraints in PCF file	N/A

PAR Operation

The following sections describe how placement and routing are performed by PAR.

Placement

The PAR module places in two stages: a constructive placement and an optimizing placement. PAR writes the NCD file after constructive placement and modifies the NCD after optimizing placement.

During constructive placement, PAR places components into sites based on factors such as constraints specified in the input file (for example, certain components must be in certain locations), the length of connections, and the available routing resources. This placement also takes into account “cost tables”, which assign weighted values to each of the relevant factors. There are 100 possible cost tables. Constructive placement continues until all components are placed. PAR writes the NCD file after constructive placement.

The optimizing placement is a fine tuning of the results of the constructive placement. Optimizing is run only at specific levels, and the number of passes may vary. PAR rewrites the NCD file after optimizing placement.

Timing-driven placement is automatically invoked if PAR finds timing constraints in the physical constraints file.

Routing

Routing is done in two stages: constructive routing and cleanup. PAR writes the NCD file only at the end of an iteration after more than 60 minutes of routing have elapsed, and it only writes out a new NCD file if the design quality improves.

During constructive routing, the router performs an iterative procedure to converge on a solution that accomplishes these goals.

- Routing the design to completion.
- Meeting your timing constraints.

If both of these goals cannot be met, the first is considered more important; that is, PAR tries to route to completion even if your timing constraints are not met.

During cleanup routing, the router takes the result of constructive routing and reroutes some connections to accomplish these goals.

- Minimizing the delays on all nets.
- Decreasing the number of routing resources used.

If both of these goals cannot be met, the first is considered more important; that is, PAR tries to route to minimize delays in preference to decreasing the number of routing resources used.

There are two types of cleanup routing you can perform—a faster cost-based cleanup routing and a more intensive delay-based cleanup routing. Cost-based cleanup runs much faster than delay-based cleanup, but delay-base cleanup usually produces a result that has faster in-circuit performance.

Timing-driven routing is automatically invoked if PAR finds timing constraints in the physical constraints file.

Note To achieve your timing constraints while routing an XC4000E/L/EX/XL design, PAR may add an additional pullup to a net at the output of a TBUF. PAR adds this pullup to the longline on which the net is routed. The pullup is added if the net contains a single pullup and the design has been completely routed, but the net containing the pullup has one or more timing errors.

Guided PAR

You can use guide files to modify your design incrementally or you can integrate your design with PCI Core guide files. The following sections describe both types of guided PAR use.

Incremental Designs

An optional guide design file can be fed into PAR. The guide file is an NCD file which is used as a template for placing and routing the input design. This is useful if minor incremental changes have been made to create a new design. To increase productivity, you can use your last design iteration as a guide design for the next design iteration, that is, your output NCD file becomes the guide design file for your next iteration of the design (see the following figure).

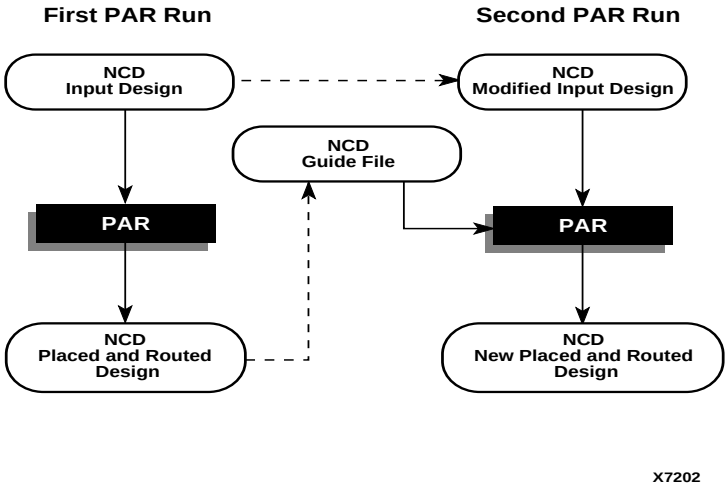


Figure 12-4 Guided PAR for Incremental Design

Two command line options control guided PAR. The –gf option specifies the NCD guide file, and the –gm option determines whether exact mode or leveraged mode is used to guide PAR.

The guide design is used as follows.

- If a component in the new design is constrained to the same location that a component is placed in the guide file, this component will be defined as matching.
- If a component in the new design has the same name as a component in the guide design, that component will match the guide component.
- If a signal in the new design has the same name as a signal in the guide design, the signal will match the guide signal.
- Any matching component in the new design will be placed in the site corresponding to the location of the matching guide component, if possible.
- Matching component's pins will be swapped to match those of the guide component with regard to matching signals, if possible.
- All of the connections between matching driver and load pins of the matching signals will have the routing information preserved from the guide file, if possible.

When PAR runs using a guide design as input, PAR first places and routes any components and signals that fulfill the matching criteria described above. Then PAR places and routes the remainder of the logic.

To place and route the remainder of the logic, PAR does the following.

- If you have selected exact guided PAR (by entering the **–gm exact** option on the PAR command line), the placement and routing of the matching logic are locked. Neither placement nor routing can be changed to accommodate the additional logic.
- If you have selected leveraged guided PAR (by entering the –**gm leverage** option on the PAR command line), PAR tries to maintain the placement and routing of the matching logic, but changes placement or routing if it is necessary in order to place and route to completion and achieve your timing constraints (if possible).

  Some cases where the leveraged mode is necessary are as follows:

  - ♦ You have added logic that makes it impossible to meet your timing constraints without changing the placement and routing in the guide design.

- You have added logic that demands a certain site or certain routing resource, and that site or routing resource is already being used in the guide design.

  As one example of this condition, in XC4000EX devices TBUFs must be routed along long lines. If you add TBUFs to an XC4000EX design but your guide design uses too many of the required long lines, you are not able to route this design to completion unless you use the leveraged option.

If you enter a –gm (guide mode) option but do not specify a guide file with the –gf option, PAR is guided by the placement and routing information in the input NCD file. Depending on whether you specify exact mode or leveraged mode, PAR locks the input NCD's existing placement and routing (exact mode), or tries to maintain the placement and routing, but modifies them in an effort to place and route to completion and achieve your timing constraints (leveraged mode).

Note For Verilog or VHDL netlist input designs, re-synthesizing modules typically cause signal and instance names in the resulting netlist to be significantly different from the netlist obtained in earlier synthesis runs. This occurs even if the source level Verilog or VHDL code only contains a small change. Because guided PAR depends on signal and component names, synthesis designs often have a low "match rate" when guided. Therefore, guided PAR is not recommended for most synthesis-based designs, although there may be cases where it could be a successful alternative technique.

PCI Cores

For the 3.1i release, you can use a guide file to add a PCI Core, which is a standard I/O interface, to your design. The PCI Core guide file must already be placed and routed. PAR only places and routes the signals that run from the PCI Core to the input NCD design; it does not place or route any portion of the PCI Core. You can also use the resulting design (PCI Core integrated with your initial design) as a guide file. However, you must then use the **exact** option for -gm, *not* **leverage**, when generating a modified design.

Guided PAR supports more precise matching of placement and routing of PCI Cores that are used as reference designs in a guide file:

- Components locked in the input design are guided by components in the reference design of a guide file in the corresponding location.
- Signals that differ only by additional loads in the input design will have the corresponding pins routed according to the reference design in the guide file.
- Guide summary information in the PAR report describes the amount of logic from the reference design that matches logic in the input design.

For detailed information about designing with PCI, refer to the Xilinx PCI web page (http://www.xilinx.com/products/logicore/pci/pcilit.htm).

Output from PAR

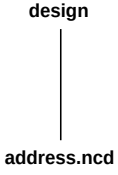
The output of PAR is a placed and routed NCD file (the output design file). In addition to the output design file, a PAR run generates a report file with a .par extension, a delay file with a .dly extension, and a pinout file with a .pad extension. The PAR file contains execution information about the place and route job as well as all constraint messages. The DLY file contains delay information about the routed nets in the design. The PAD file lists IOBs (Input/Output Blocks) on the chip and the primary pins associated with the IOBs.

If the options that you specify when running PAR are options that produce a single output design file, your output is the output design file, a PAR file, a DLY file, and a PAD file. The PAR file, the DLY file, and the PAD file all have the same root name as the output design file.

If you run multiple iterations of placement and routing, you produce an output design file, a PAR file, a DLY file, and a PAD file for each iteration. Consequently, when you run multiple iterations you have to specify a directory in which to place these files.

As the command is performed, PAR records a summary of all placement and routing iterations in one PAR file at the same level as the directory you specified, then places the output files (in NCD format) in the specified directory. Also, a PAR file, a DLY file, and a PAD file are created for each NCD file, describing in detail each individual iteration.

For example, suppose you have a directory named design with a design file called address.ncd, as shown in the following figure.



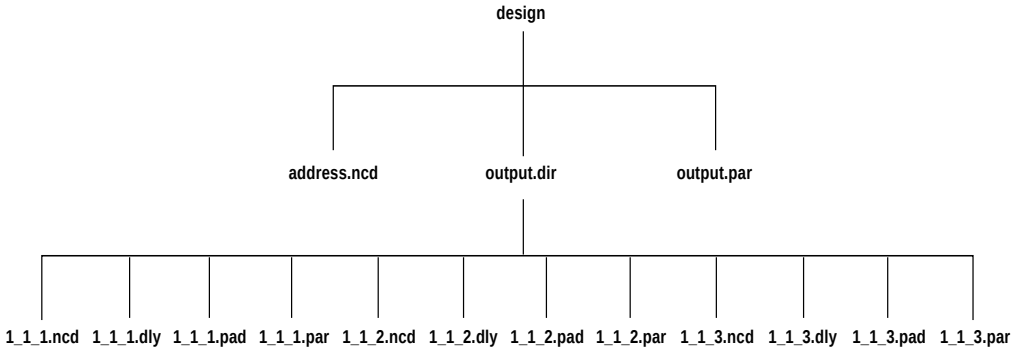
X7231

Suppose you run three iterations of place and route, using a different cost table entry each time (cost tables are explained in the "Placement" section) and specify that the resulting output be put into a directory called output.dir. The actual command would be

```
par –n 3 -l 1 address.ncd output.dir
```

–n 3 is the number of iterations you want to run, **–l 1** sets the placement effort level, **address.ncd** is your input design file, and **output.dir** is the name of the directory in which you want to place the results of the PAR run.

The files resulting from the command are shown in the following figure.



The naming convention for the files, which may contain placement and routing information in varying degrees of completion, is *placer_level_router_level_table.file_extension*.

In the sample above, the effort level and cost table entries start at 1 (the default value). The PAR, DLY, and PAD files are described in the following sections. When you run multiple iterations, you get a summary PAR report file like the one shown below.

Note The PAR Report is formatted for viewing in a monospace (non-proportional) font. If the text editor you use for viewing the report uses a proportional font, the columns in the report do not line up correctly.

```
PAR: Xilinx Place And Route 3.1i.
Copyright (c) 1995-2000 Xilinx, Inc.  All rights reserved.

par -ol 3 -n 5 -i 20 main_pcb.ncd main_pcb.pcf


Constraints file: main_pcb.pcf.

Level/      Design    Timing    Number    Run         NCD
Cost [ncd]  Score     Score     Unrouted  Time        Status
----------  -------  --------  -------    ----       ------------
3_3_1   *   716         0         0       01:02       Complete
3_3_5   *   724         0         0       01:08       Complete
3_3_2   *   730         0         0       01:05       Complete

3_3_4   *   812         0         0       01:06       Complete
3_3_3   *   827         0         0       01:13       Complete

* : Design saved.

par done!
```

At the top of the summary PAR file is information regarding the software level, copyright information, and the date and time of the run. Directly below that is the command line used to run PAR, followed by the name of any physical constraints file used.

The body of the report consists of the following columns.

Level / Cost [ncd]—indicates the effort level (1–5) at which PAR is run. In the sample above, 3_3_4 indicates placer level 3, router level 3, and the fourth cost table used.

Design Score—see “Incremental Designs” section.

Timing Score—see “Incremental Designs” section.

Number Unrouted—indicates the number of unrouted nets in the design.

Run Time—the time required to complete the job in minutes and seconds.

NCD Status—describes the state of the output NCD file generated by the PAR run. Possible values for this column are

- Complete—an NCD file has been generated by a full PAR run.
- ^C Checkpoint—initiated by the user, the PAR run was stopped at one of the PAR checkpoints. PAR produced an NCD file, but all iterations may not have been completed.
- Checkpoint—the PAR run was stopped at one of the PAR checkpoints, not because of user intervention but because of some unknown reason.
- No NCD—the PAR job was stopped prematurely and the NCD file was not checkpointed.

Intermediate Failing Timespec Summary

PAR generates an intermediate failing timespec summary only in the routing phase. The summary name is design_name.itr.

The router creates this summary after an iteration not during an iteration. If interrupted during normal operation of an iteration (for example, CTRL-C), you are prompted with the following options if a time specification has failed:

```
CNTRL-C interrupt detected.
Please choose one of the following options:

1. Continue processing and ignore the interrupt.
2. Normal program exit at next check point.
   This will result in saving the best results so
   far,
   after concluding current processing.
3. Exit program immediately.
4. Display Failing Timespec Summary.
5. Cancel the current job and move to the next one
   at
    the next check point.
Enter choice -->
```

If you select 3. PAR exits. If you select 4, PAR displays the contents of the ITR file on the screen and resumes execution. (Option 5 allows you to terminate jobs that use the -n option for multiple iterations.) If

Options 4 and 5 are not applicable, the following messages displays for those options on a CTRL C instead of the ones shown previously.

```
4. Display Failing Timespec Summary.
    (Not applicable: Data not available)
5. Cancel the current job and move to the next one
   at
    the next check point.
    (Not applicable: Not a multi-run job.)
```

Following is a sample ITR report.

```
Asterisk (*) preceding a constraint indicates it was not met.
------------------------------------------------------------------
  Constraint                          | Requested |Actual | Logic
                                      |           |       |Levels
------------------------------------------------------------------
* OFFSET = OUT 15 nS  AFTER COMP "ck1_i"| 15.000ns|15.800ns| 5
------------------------------------------------------------------

1 constraint not met.
```

PAR creates an intermediate failing timespec summary generated from the end of the previous iteration. If the interrupt occurred during the first iteration, no intermediate summary is created.

The Place and Route (PAR) Report File

The place and route (PAR) report file contains execution information about the PAR command run. The file shows the steps taken as the program converges on a placement and routing solution. A sample PAR file is shown following.

Note The PAR Report is formatted for viewing in a monospace (non-proportional) font. If the text editor you use for viewing the report uses a proportional font, the columns in the report do not line up correctly.

```
Release 3.1i - Par D.18
Wed Mar  8 11:12:50 2000

par -w xcvsink.ncd routed

Constraints file: xcvsink.pcf
```

```
Loading device database for application par from file "xcvsink.ncd".
  "Synopsys_edif" is an NCD, version 2.28, device xcv50, package
pq240,speed -6
Loading device for application par from file 'v50.nph' in
environment /build/bcxfndry/D.18/rtf.
Device speed data version:  PRELIMINARY 1.99 2000-03-02.

Device utilization summary:

    Number of External GCLKIOBs          1 out of 4      25%
    Number of External IOBs              8 out of 166     4%

    Number of SLICEs                     5 out of 768     1%

    Number of GCLKs                      1 out of 4      25%

Overall effort level (-ol):   2 (default)
Placer effort level (-pl):    2 (default)
Placer cost table entry (-t): 1
Router effort level (-rl):    2 (default)

Starting initial Placement phase. REAL time: 4 secs
Finished initial Placement phase. REAL time: 4 secs
Starting the placer. REAL time: 4 secs
Placement pass 1 .
Placer score = 660
Optimizing ...
Placer score = 645
Starting IO Improvement.    REAL time: 4 secs
                            CPU  time: 2 secs
Placer score = 645
Finished IO Improvement.    REAL time: 4 secs
                            CPU  time: 2 secs

Placer completed in real time: 4 secs

Writing design to file "routed.ncd".
Total REAL time to Placer completion: 4 secs
```

```
Total CPU time to Placer completion: 2 secs

PAR Placer Norm: NPU  time: 6 secs
0 connection(s) routed; 26 unrouted.
Starting router resource preassignment
Completed router resource preassignment.  REAL time: 4 secs
                                          CPU  time: 2 secs
                                          NPU  time: 7 secs
Starting iterative routing.
Routing active signals.
.
End of iteration 1
26 successful; 0 unrouted; (0)  REAL time: 5 secs
                                CPU  time: 3 secs
                                NPU  time: 7 secs
Constraints are met.
Total REAL time: 5 secs
      NPU  time: 7 secs
Total CPU  time: 3 secs
      NPU  time: 7 secs
End of route.  26 routed (100.00%); 0 unrouted.
No errors found.
Completely routed.

This design was run without timing constraints.  It is likely that
much better circuit performance can be obtained by trying either or
both of the following:

  - Enabling the Delay Based Cleanup router pass, if not already
enabled
  - Supplying timing constraints in the input design

Total REAL time to Router completion: 5 secs
Total CPU time to Router completion: 3 secs

Generating PAR statistics.

  The Delay Summary Report

  The Score for this design is: 122
```

```
The Number of signals not completely routed for this design is: 0

  The Average Connection Delay for this design is:        0.961 ns
  The Maximum Pin Delay is:                               1.842 ns
  The Average Connection Delay on the 10 Worst Nets is:   1.303 ns

  Listing Pin Delays by value: (ns)
d < 1.00   < d < 2.00  < d < 3.00  < d < 4.00  < d < 5.00  d >= 5.00
---------  ---------  ---------  ---------  ---------  ---------
          9         17          0          0          0          0
Writing design to file "routed.ncd".

All signals are completely routed.

Total REAL time to PAR completion: 6 secs
Total CPU time to PAR completion: 3 secs
PAR Norm: NPU  time: 8 secs

=========================================
Timing wizard resettargets history.

Entry     TWiters   TWtime
-----     -------   ------
=========================================
Time spent in Timing analysis. CPU: 0 secs
Time spent in Timing analysis. NPU: 0 secs
=========================================

Placement: Completed - No errors found.
Routing: Completed - No errors found.

PAR done.
```

Sometimes the design is completely routed, but the router continues to route in the attempt to meet timing constraints.

Note that in the sample PAR file above, in the “starting iterative routing” section, after the end of iteration 1, there is a figure in parentheses (0). This represents the timing score for the design (not to be confused with the PAR score) at the end of the particular iteration.

This figure appears in the PAR file only when timing constraints have been specified in a PCF file. When the timing score is 0 (as it is in this example after iteration 1), this means that all timing constraints have been met. This score (0) also appears at the end of the delay report section of the PAR file.

The timing score at the end of the "starting iterative routing" section may not agree with the timing score in the Delay Summary Report. This can occur if a MAXSKEW constraint is scored and not met.

Had the design been completely routed but failed to meet all timing constraints, the score would have been a figure other than 0. A non-zero number would appear at the end of the delay report section. This tells you immediately whether your timing constraints have been met. It is possible that the timing score shown in parentheses at the top of the file may be different from the one shown in the delay summary section of the file. The score shown in the delay summary section is always the correct one.

The last section of the PAR file contains a summary of the delay information for the routed design. The DLY (delay) file produced by the PAR run contains more detailed timing information. The DLY file is discussed in the following section.

If you specify a command option that produces multiple output design files, there is a PAR file indicating all of the place and route iterations performed, and individual PAR files describing placement and routing for each design file produced.

Note In PAR reporting, a tilde (~) preceding a delay value indicates that the delay value is approximate. Values with the tilde cannot be calculated exactly because of excessive delays, resistance, or capacitance on the net. You can use the PENALIZE TILDE constraint to penalize these delays by a specified percentage (see the "TRACE" chapter and the "Attributes, Constraints, and Carry Logic" chapter of the *Libraries Guide* for a description of the PENALIZE TILDE constraint).

Some notes about the entries in the PAR file.

- The Placer score is a rating of the relative "cost" of a placement. A lower score indicates a better (that is, less "costly") placement.
- In the Delay Summary Report section of the PAR report file where average delays are listed (beginning with THE AVERAGE CONNECTION DELAY for this design), there are two columns of

figures. The first column gives the actual averages for the design. The figures in the second column, which are enclosed by parentheses, indicate the averages after the imposition of a tilde penalty.

- The Score For This Design is a rating of the routed design. The score is discussed in the “Scoring the Routed Design” section
- Timing score is always 0 (zero) if all timing constraints have been met. If not, the figure is other than 0.

For the Virtex/-E/-II and Spartan-II devices, if more than one SelectIO standard is used, an additional section on Select IO utilization and usage summary is added to the PAR file. This section shows details for the different IO banks. It shows the IO standard, the output reference voltage (VCCO)] for the bank, the input reference voltage (VREF) for the bank, the PAD and Pin names. In addition, the section gives a summary for each bank with the number of pads being used, the voltages of the VREFs, and the VCCOs. A sample Select IO utilization and Usage Summary of the PAR file follows.

```
Select IO Utilization and Usage Summary
_______________________________________
NR - means Not Required.
Each Group of a specific Standard is listed.
IO standard (LVTTL Vref=NR Vcco=3.30) occupies 45 pads.
IO standard (CTT Vref=1.50 Vcco=3.30) occupies 8 pads.
IO standard (SSTL3_I Vref=0.90 Vcco=3.30) occupies 12 pads.
Bank Summary
____________
NR - means Not Required
Bank 0 has 20 pads and is 80% utilized.
Vref should be set to NR volts.
Vcco should be set to 3.30 volts.
      Name              IO    Select  StdVref Vcco Pad  Pin
      ----              ----------------------------------
      bidir<7           IO    LVTTL     NR  3.30  PAD2  P238
      bidir<6>          IO    LVTTL     NR  3.30  PAD3  P237
      bidir<3>          IO    LVTTL     NR  3.30  PAD8  P231
      bidir<1>          IO    LVTTL     NR  3.30  PAD10 P230
      b<10>             I     LVTTL     NR        PAD11 P229
      .
      .
      .
      b<7>              I     LVTTL     NR        PAD17 P221
      a<10>             I     LVTTL     NR        PAD18 P220
Bank 1 has 22 pads and is 13% utilized.
Vref should be set to NR volts.
     Name                IO  Select Std Vref   Vcco   Pad   Pin
     ----                --  ---------- ------ ------ ------ ------
     .
     .
     .
Bank 7 has 21 pads and is 38% utilized.
Vref should be set to 0.90 volts.
Vref sites in this bank cannot be used for user IOBs.
Vcco should be set to 3.30 volts.
    Name            IO  Select   StdVref Vcco Pad   Pin
    ----            ------------------------------------
    bidir<11>       IO  SSTL3_I   0.90  3.30  PAD169 P28
    bidir<8>        IO  SSTL3_I   0.90  3.30  PAD170 P27
    bidir<9>        IO  SSTL3_I   0.90  3.30  PAD172 P25
    bidir<10>       IO  SSTL3_I   0.90  3.30  PAD173 P24
```

```
    c<9>            O   CTT             3.30  PAD181 P13
    c<10>           O   CTT             3.30  PAD187 P7
    c<7             O   LVTTL           3.30  PAD190 P4
    c<8>            O   CTT             3.30  PAD191 P3
Total REAL time to Placer completion: 40 secs
Total CPU time to Placer completion: 31 secs
```

For guided par, the PAR report displays summary information describing the total amount and percentage of components and signals in the input design guided by the reference design. The report also displays the total / percentage of components and signals from the reference design (guide file) that were used to guide the input design. See the "Guide Reporting" section.

The Delay (DLY) File

The delay file is output by each PAR run and is placed in the directory with the NCD output of the design file and the PAR file. The delay file contains delay information for each net in the design and includes the following:

- A listing of the 20 nets with the longest delays., In a DLY file, maximum delays are preceded by a tilde, indicating that the delay shown is only approximate. Following each tilde delay is a figure in parentheses. This figure represents the approximate delay with a certain percentage automatically added to it (a "worst case" situation) when specified by the user in the physical constraints (PCF) file. When the Xilinx Development System's timing analysis software looks at the delays, it uses the value in parentheses rather than the approximate value represented by the tilde. For more information on the PENALIZE TILDE constraint, see the "TRACE" chapter in this manual and the "Attributes, Constraints, and Carry Logic" chapter of the *Libraries Guide*.
- A delay analysis for each net, including the net name, followed by the driver pin and the load pin(s).

The following is a portion of a delay file. If this were a complete file, it would show the load delays for all nets in the design.

Note The Delay Report is formatted for viewing in a monospace (non-proportional) font. If the text editor you use for viewing the

report uses a proportional font, the columns in the report do not line up correctly.

```
Wed Mar  8 11:12:56 2000

File: routed.dly

  The 20 Worst Net Delays are:
------------------------------
| Max Delay (ns)  | Netname   |
------------------------------
   1.842            n112
   1.770            n107
   1.596            n114
   1.590            n109
   1.590            n110
   1.573            n111
   1.361            n108
   1.361            n113
   0.345            n58
   0.006            U15/P
   0.000            add_18/INC_AND<4>
   0.000            add_18/INC_AND<6>
   0.000            add_18/INC_AND<2>

--------------------------------------------------------------------
                        Net Delays
--------------------------------------------------------------------
U15/P
   clk.GCLKOUT
         0.006  U15/U2.IN

add_18/INC_AND<2>
   n113.COUT
         0.000  n111.CIN

add_18/INC_AND<4>
   n111.COUT
         0.000  n109.CIN

add_18/INC_AND<6>
```

```
   n109.COUT
         0.000  n107.CIN

n107
   n107.XQ
         1.770  incout<7>.O
         1.168  n107.F2

n108
   n109.YQ
         1.361  incout<6>.O
         1.038  n109.G4

n109
   n109.XQ
         1.590  incout<5>.O
         1.323  n109.F1

n110
   n111.YQ
         1.590  incout<4>.O
         1.038  n111.G4

n111
   n111.XQ
         1.573  incout<3>.O
         1.168  n111.F2

n112
   n113.YQ
         1.842  incout<2>.O
         1.082  n113.G3

n113
   n113.XQ
         1.361  incout<1>.O
         1.168  n113.F2

n114
   n114.YQ
```

```
         1.461  incout<0>.O
         1.596  n113.BX
         1.215  n114.BY

n58
   U15/U2.OUT
         0.309  n113.CLK
         0.345  n111.CLK
         0.345  n109.CLK
         0.345  n107.CLK
         0.302  n114.CLK
```

The PAD File

The PAD file contains a listing of all IOBs used in the design and their associated pads. The file specifies connections to device pins (with a P prefix).

The PAD file is divided into three sections.

- The first section lists the component name in the first column. The second column of this section lists the designations of the device pins.
- The second section lists the pin number in the first column, the component name in the second column, and any constraints assigned to the component in the third column.
- The third section lists the pinouts in the form of constraints. These constraints can be cut and pasted into a PCF file as constraints for later PAR runs.

For 3.1i, the PAD file reports all dual-purpose pins that are used during configuration as well during normal operation.

For the Virtex/-E/-II or Spartan-II devices, when SelectIOs are used, the PAD file also contains details of the pads that must be used for the input reference voltage (VREF), and those that must be used for the output reference voltage (VCCO). For the VREF pads, their location and the value of the input reference voltage is shown. A sample Virtex PAD file follows.

Note The PAD Report is formatted for viewing in a monospace (non-proportional) font. If the text editor you use for viewing the report

uses a proportional font, the columns in the report do not line up correctly.

```
Release 3.1i - Par D-18
Wed Mar  8 09:54:08 2000
Xilinx PAD Specification File
****************************
Input file:       mapped.ncd
Output file:      t.ncd
Part type:        xc2v250
Speed grade:      -6
Package:          fg256
Pinout by Pin Name:
+-----------------------------------------------+----------+----------------------------
|Pin Name| Direction | Pin    | IO       |Drive    | Slew | Resistor |
|        |           | Number | Standard |Strength | Rate |          |
+-------------------------------------------------------+------------+---
| clk    |           |                                  | INPUT    | L5     | LVDS     |
| clkb                                                  | INPUT    | K1     |          |
|        |           |
| d                                                     | INPUT    | M1     | LVDS     |
|        |           |
| db                                                    | INPUT    | M2     |          |
|        |           |
| io                                                    | OUTPUT   | M3     | LVDS     |
|        |           |
| iob                                                   | OUTPUT   | M4     |          |
|        |           |
| o                                                     | OUTPUT   | L3     | LVDS     |
|        |           |
| ob                                                    | OUTPUT   | L4     |          |
|        |           |
| to                                                    | OUTPUT   | L1     | LVDS     |
|        |           |
| tob (VREF)                                            | OUTPUT   | L2     |          |
|        |           |
+-----------------------------------------------+----------+---+
Dedicated or Special Pin Name:
+-------------------------------------------------------------+----
|               Dedicated or Special Pin Name                 | Pin    | Voltage |
|                                                             | Number |         |
+-------------------------------------------------------------+---+
| CCLK                                                        | T14    |         |
| DONE                                                        | T15    |         |
| DXN                                                         | C15    |         |
| DXP                                                         | C16    |         |
| GND                                                         | G10    |         |
.
.
| GND                                                         | L10    |         |
| HSWAP_EN                                                    | A3     |         |
| M0                                                          | D16    |         |
| M1                                                          | D15    |         |
| M2                                                          | D14    |         |
| NC2                                                         | A4     |         |
| PROG                                                        | A2     |         |
| PWRDWN_B                                                    | C2     |         |
| TCK                                                         | N1     |         |
| TDI                                                         | N3     |         |
| TDO                                                         | P15    |         |
| TMS                                                         | N2     |         |
```

```
| VCCI                                                    | R1     |         |
.
.
.
| VCCO_4                                                  | M9     | na      |
| VCCO_4                                                  | L9     | na      |
| VCCO_5                                                  | L8     | na      |
| VCCO_5                                                  | M8     | na      |
| VCCO_6                                                  | J5     | na      |
| VCCO_6                                                  | J6     | na      |
| VCCO_7                                                  | H5     | na      |
| VCCO_7                                                  | H6     | na      |
| VPP                                                     | B3     |         |
| VREF                                                    | A8     |         |
.
.
| VREF                                                    | G3     |         |
| VREF                                                    | F15    |         |
| VREF                                                    | F2     |         |
| VREF                                                    | E3     |         |
+---------------------------------------------------------+----
Pinout by Pin Number:
+-------------+-----------------------------------+-----------+--
|  Pin Number |             Pin Name              | Direction | Constraint |
+-------------+-----------------------------------+-----------+------------+
| A1          | (GND)                             |           |            |
| A2          | (PROG)                            |           |            |
| A3          | (HSWAP_EN)                        |           |            |
| A4          | (NC2)                             |           |            |
| A5          | ---                               | UNUSED    |            |
.
.
.
| C11         | ---                               | UNUSED    |            |
| C12         | ---                               | UNUSED    |            |
| C13         | ---                               | UNUSED    |            |
.
.
| E3          | (VREF)                            |           |            |
| E4          | ---                               | UNUSED    |            |
| E5          | (VCCINT)                          |           |            |
| E6          | (VREF)                            |           |            |
| E7          | ---                               | UNUSED    |            |
| E8          | (VCCO_0)                          |           |            |
| E9          | (VCCO_1)                          |           |            |
| E10         | ---                               | UNUSED    |            |
| E11         | (VREF)                            |           |            |
| E12         | (VCCINT)                          |           |            |
| E13         | ---                               | UNUSED    |            |
| E14         | ---                               | UNUSED    |            |
| E15         | ---                               | UNUSED    |            |
| E16         | ---                               | UNUSED    |            |
| F1          | ---                               | UNUSED    |            |
| F2          | (VREF)                            |           |            |
| F3          | ---                               | UNUSED    |            |
| F4          | ---                               | UNUSED    |            |
| F5          | ---                               | UNUSED    |            |
| F6          | (GND)                             |           |            |
| F7          | (GND)                             |           |            |
| F8          | (VCCO_0)                          |           |            |
| F9          | (VCCO_1)                          |           |            |
| F10         | (GND)                             |           |            |
| F11         | (GND)                             |           |            |
```

```
.
.
| G10          | (GND)                          |           |            |
| G11          | (GND)                          |           |            |
| G12          | ---                            | UNUSED    |            |
| G13          | ---                            | UNUSED    |            |
| G14          | (VREF)                         |           |            |
| G15          | ---                            | UNUSED    |            |
| G16          | ---                            | UNUSED    |            |
| H1           | ---                            | UNUSED    |            |
| H2           | ---                            | UNUSED    |            |
| H3           | ---                            | UNUSED    |            |
| H4           | ---                            | UNUSED    |            |
| H5           | (VCCO_7)                       |           |            |
| H6           | (VCCO_7)                       |           |            |
| H7           | (GND)                          |           |            |
| H8           | (GND)                          |           |            |
| H9           | (GND)                          |           |            |
| H10          | (GND)                          |           |            |
| H11          | (VCCO_2)                       |           |            |
| H12          | (VCCO_2)                       |           |            |
| H13          | ---                            | UNUSED    |            |
| H14          | ---                            | UNUSED    |            |
| H15          | (VREF)                         |           |            |
| H16          | ---                            | UNUSED    |            |
| J1           | ---                            | UNUSED    |            |
.
.
.
| J14          | ---                            | UNUSED    |            |
| J15          | ---                            | UNUSED    |            |
| J16          | ---                            | UNUSED    |            |
| K1           | clkb                           | INPUT     |            |
| K2           | ---                            | UNUSED    |            |
| K3           | (VREF)                         |           |            |
| K4           | ---                            | UNUSED    |            |
| K5           | ---                            | UNUSED    |            |
| K6           | (GND)                          |           |            |
| K7           | (GND)                          |           |            |
| K8           | (GND)                          |           |            |
| K9           | (GND)                          |           |            |
| K10          | (GND)                          |           |            |
| K11          | (GND)                          |           |            |
| K12          | ---                            | UNUSED    |            |
| K13          | ---                            | UNUSED    |            |
.
.
.
| L16          | ---                            | UNUSED    |            |
| M1           | d                              | INPUT     |            |
| M2           | db                             | INPUT     |            |
| M3           | io                             | OUTPUT    |            |
| M4           | iob                            | OUTPUT    |            |
| M5           | (VCCINT)                       |           |            |
| M6           | (VREF)                         |           |            |
| M7           | ---                            | UNUSED    |            |
| M8           | (VCCO_5)                       |           |            |
| M9           | (VCCO_4)                       |           |            |
| M10          | ---                            | UNUSED    |            |
| M11          | (VREF)                         |           |            |
| M12          | (VCCINT)                       |           |            |
| M13          | ---                            | UNUSED    |            |
.
```

```
.
.
| P6           | ---                                | UNUSED    |            |
| P7           | ---                                | UNUSED    |            |
| P8           | ---                                | UNUSED    |            |
| P9           | ---                                | UNUSED    |            |
| P10          | ---                                | UNUSED    |            |
| P11          | ---                                | UNUSED    |            |
| P12          | (VREF)                             |           |            |
| P13          | ---                                | UNUSED    |            |
| P14          | (VCCINT)                           |           |            |
| P15          | (TDO)                              |           |            |
| P16          | ---                                | UNUSED    |            |
| R1           | (VCCI)                             |           |            |
| R2           | (GND)                              |           |            |
| R3           | ---                                | UNUSED    |            |
| R4           | (VREF)                             |           |            |
.
.
.
| R15          | (GND)                              |           |            |
| R16          | (VCCI)                             |           |            |
| T1           | (GND)                              |           |            |
| T2           | ---                                | UNUSED    |            |
| T3           | ---                                | UNUSED    |            |
| T4           | ---                                | UNUSED    |            |
| T5           | ---                                | UNUSED    |            |
| T6           | ---                                | UNUSED    |            |
| T7           | ---                                | UNUSED    |            |
| T8           | ---                                | UNUSED    |            |
| T9           | (VREF)                             |           |            |
| T10          | ---                                | UNUSED    |            |
| T11          | ---                                | UNUSED    |            |
| T12          | ---                                | UNUSED    |            |
| T13          | ---                                | UNUSED    |            |
| T14          | (CCLK)                             |           |            |
| T15          | (DONE)                             |           |            |
| T16          | (GND)                              |           |            |
+--------------+------------------------------------+-----------+--
#
# To preserve the pinout above for future design iterations,
# simply run "Lock Pins..." from the Design Manager's Design
# menu, or invoke PIN2UCF from the command line. The location constraints
# above will be written into your specified UCF file. (The constraints
# listed below are in PCF format and cannot be directly used in the UCF file).
#
COMP "clk" LOCATE = SITE "L5" ;
COMP "clkb" LOCATE = SITE "K1" ;
COMP "d" LOCATE = SITE "M1" ;
COMP "db" LOCATE = SITE "M2" ;
COMP "io" LOCATE = SITE "M3" ;
COMP "iob" LOCATE = SITE "M4" ;
COMP "o" LOCATE = SITE "L3" ;
COMP "ob" LOCATE = SITE "L4" ;
COMP "to" LOCATE = SITE "L1" ;
COMP "tob" LOCATE = SITE "L2" ;
#
```

Guide Reporting

This report, which is included in the PAR report file, is generated when using the -gf option. The report describes the criteria used to select each component and signal used to guide the design. It may also enumerate the criteria used to reject some subset of the components and signals that were eliminated as candidates.

Following is an example of guide file information that displays in the PAR file.

```
Release 3.1i - Par D.18
Wed Mar  8 14:14:17 2000
Constraints file: mapped.pcf
Loading device database for application par from file "mapped.ncd".
   "lvds" is an NCD, version 2.32, device xc2v250, package fg256,
speed -6
Loading device for application par from file '2v250.nph' in
environment
Device speed data version:  DEVELOPMENT 1.39b 2000-03-06.
Starting Guide File Processing.
Loading device database for application par from file "jay.ncd".
   "lvds" is an NCD, version 2.32, device xc2v250, package fg256,
speed -6
Finished Guide File Processing.
Guide Summary Report:
   Guide file: "jay.ncd"    Guide mode: "exact"
   Guide File Components matched:              12 out of   12  100%
   Placed Guide File Components:               12 out of   12  100%
   Guide File Signals matched:                 22 out of   22  100%
   Design Totals:
   Design Components matched:                  12 out of   12  100%
   Designs Components matching Placement:      12 out of   12  100%
   Design Signals matched:                     22 out of   22  100%
For a detailed guide report refer to the "guided.grf" file.
Device utilization summary:
   Number of External DIFFMs           5 out of 86      5%
   Number of External DIFFSs           5 out of 86      5%
   Number of SLICEs                    1 out of 1536    1%
   Number of BUFGMUXs                  1 out of 16      6%
Overall effort level (-ol):   2 (default)
Placer effort level (-pl):    2 (default)
Placer cost table entry (-t): 1
```

```
Router effort level (-rl):    2 (default)
Starting Clock Logic Placement.  REAL time: 15 secs
Finished Clock Logic Placement.  REAL time: 16 secs
Starting clustering phase.       REAL time: 16 secs
Finished clustering phase.  REAL time: 16 secs
Starting Directed Placer.   REAL time: 17 secs
Placement pass 1 .
Placer score = 1920
Placer score = 1920
Finished Directed Placer.    REAL time: 17 secs
Starting Optimizing Placer.  REAL time: 17 secs
Optimizing
Swapped 0 comps.
Xilinx Placer [1]   1920     REAL time: 17 secs
Finished Optimizing Placer.  REAL time: 17 secs
Writing design to file "guided.ncd".
Total REAL time to Placer completion:   18 secs
Total CPU time to Placer completion:     5 secs
14 connection(s) routed; 0 unrouted active, 1 unrouted PWR/GND.
Starting router resource preassignment
Completed router resource preassignment. REAL time: 19 secs
                                         CPU  time: 6 secs
                                         NPU  time: 15 secs
End of iteration 1
14 successful; 0 unrouted active,
   1 unrouted PWR/GND; (0) REAL time: 20 secs
                           CPU  time: 6 secs
                           NPU  time: 15 secs
Constraints are met.
Routing PWR/GND nets.
Power and ground nets completely routed.
Total REAL time: 20 secs
Total CPU  time: 6 secs
End of route.  15 routed (100.00%); 0 unrouted.
No errors found.
Completely routed.
Total REAL time to Router completion: 22 secs
Total CPU time to Router completion: 6 secs
Generating PAR statistics.
   The Delay Summary Report
   The Score for this design is: 86
The Number of signals not completely routed for this design is: 0
```

```
   The Average Connection Delay for this design is:        0.745 ns
   The Maximum Pin Delay is:                               4.206 ns
   The Average Connection Delay on the 10 Worst Nets is:   0.619 ns
   Listing Pin Delays by value: (ns)
    d < 1.00   < d < 2.00  < d < 3.00  < d < 4.00  < d < 5.00  d >= 5.00
   ---------   ---------   ---------   ---------   ---------   -------
         13          1           0           0           1           0
Writing design to file "guided.ncd".
All signals are completely routed.
Total REAL time to PAR completion: 25 secs
Total CPU time to PAR completion: 7 secs
Placement: Completed - No errors found.
Routing: Completed - No errors found.
PAR done.
```

Scoring the Routed Design

The SCORE FOR THIS DESIGN is a rating of the routed design. The PAR file (a portion of which is shown below) shows the total score as well as the individual factors making up the score. The score takes the following factors into account (weighted by their relative importance).

- The number of unrouted nets (unr)
- The number of timing constraints not met (ncst)
- The amount (expressed in ns) that the timing constraints were not met (acst)
- Maximum delay on a net with a weight greater than 3
- Net weights or priorities
- The average of all of the maximum delays on all nets (av)
- The average of the maximum delays for the ten highest delay nets (10w)

The lower the score, the better the result.

The formula that produces the score is

5000*unr + 1000*ncst + 20*acst + (delay*weight)*0.2 + av*100 + 10w*20

The score in the PAR Report is shown following.

Note The PAR file is formatted for viewing in a monospace (non-proportional) font. If the text editor you use for viewing the report uses a proportional font, the columns in the report do not line up correctly.

```
The Delay Summary Report
    The SCORE FOR THIS DESIGN is: 230

The NUMBER OF SIGNALS NOT COMPLETELY ROUTED for this design is: 0
    The AVERAGE CONNECTION DELAY for this design is: 1.735
    The MAXIMUM PIN DELAY IS:                        4.603
    The AVERAGE CONNECTION DELAY on the 10 WORST NETS is:2.837
    Listing Pin Delays by value: (nsec)
    d < 1.00  d 2.00   >d < 3.00 <d < 4.00 <d < 5.00 d >= 5.00
    --------- -------- -------     -----      ------      ----
          55      127      153         2           6         0

Timing score: 0
```

When a design has been routed to your satisfaction, you can use BitGen to produce a bitstream file.

Turns Engine (PAR Multi-Tasking Option)

This Xilinx Development System option allows you to use multiple systems (nodes) that are networked together for a multi-run PAR job, significantly reducing the total amount of time to completion. You can specify multi-tasking from the UNIX command line.

Turns Engine Overview

Before the Turns Engine was developed for the Xilinx Development System, PAR could only run multiple jobs in a linear way. The total time required to complete PAR was equal to the sum of the times that it took for each of the PAR jobs to run. This is illustrated by the following PAR command.

```
par -l 5 -n 10 -i 10 -c 1 mydesign.ncd output.dir
```

The above tells PAR to run 10 place and route passes (**-n 10**) at effort level 5 (**-l 5**), a maximum of 10 router passes (**-i 10**), and one cost-based cleanup pass (c 1). It runs each of the 10 jobs consecutively, generating an output NCD file for each job, i.e., output.dir/5_5_1.ncd, output.dir/5_5_2.ncd, etc. If each job takes approximately one hour, then the run takes approximately 10 hours.

Suppose, however, that you have five nodes available. The Turns Engine allows you to use all five nodes at the same time, dramatically reducing the time required for all ten jobs. To do this you must first generate a file containing a list of the node names, one per line as in the following example.

Note A pound sign (#) in the example indicates a comment.

```
# NODE names

jupiter             #Fred's node
mars                #Harry's node
mercury             #Betty's node
neptune             #Pam's node
pluto               #Mickey's node
```

Now run the job from the command line as follows.

```
par -m nodefile_name -l 5 -n 10 -i 10 -c 1 myde-
sign.ncd output.dir
```

nodefile_name is the name of the node file you created.

This runs the following jobs on the nodes specified.

jupiter: par -l 5 -i 10 -c 1 mydesign.ncd output.dir/5_5_1.ncd
mars: par -l 5 -i 10 -c 1 mydesign.ncd output.dir/5_5_2.ncd
mercury: par -l 5 -i 10 -c 1 mydesign.ncd output.dir/5_5_3.ncd
neptune: par -l 5 -i 10 -c 1 mydesign.ncd output.dir/5_5_4.ncd
pluto: par -l 5 -i 10 -c 1 mydesign.ncd output.dir/5_5_5.ncd

As the jobs finish, the remaining jobs are started on the nodes until all 10 jobs are complete. Since each job takes approximately one hour, all 10 jobs complete in approximately two hours.

Note You cannot judge the relative benefits of multiple placements by running the Turns Engine with options that generate multiple placements but do not route any of the placed designs (the –r PAR option specifies "no routing"). The design score you receive is the same for each placement. To get some indication of the quality of the placed designs, run at least one routing iteration (–i 1) on each placed design.

Turns Engine Input Files

The following are the input files to the Turns Engine.

- NCD File—A mapped design.
- Nodelist file—A user-created ASCII file listing workstation names. A sample nodelist file is shown below.

```
# This is a comment
# Note: machines are accessed by Turns Engine
# from top to bottom


# Sparc 20 machines running Solaris
kirk
spock
mccoy
krusher
janeway
picard
# Sparc 10 machines running SunOS
michael
jermaine
marlon
tito
jackie

# HPs running HP-UX
william
george
ronald
```

```
jimmy
gerald
```

Turns Engine NCD Output File

The naming convention for the NCD file, which may contain placement and routing information in varying degrees of completion, is placer_level_router_level_table.ncd. If any of these elements are not used, they are replaced by an 'x'. For example, for the first design file being run with the options -n 5 -t 16 -rl 4 -pl 2, the NCD output file name would be 2_4_16.ncd. The second file would be named 2_4_17.ncd. For the first design file being run with the options -n 5 -t 16 -r -pl 2, the NCD output file name would be 2_x_16.ncd. The second file would be named 2_x_17.ncd.

Homogeneous and Heterogeneous Networks

The Turns Engine can run on the following networks.

- Homogenous networks—All SunOS, all Solaris, or all HP-UX.
- Heterogeneous networks—A mix of SunOS, Solaris, and HP-UX. You must have the Xilinx software and a license for each platform on which you intend to run. See the sample .cshrc file below to set up the environment variables. This is possible because the nodes read their environment variables from the .cshrc file; they do not receive them from the launching node.

Limitations

The following limitations apply to the Turns Engine.

- The Turns Engine can operate only on Xilinx FPGA families. It cannot operate on CPLDs.
- The Turns Engine can only operate on UNIX workstations.
- Each run targets the same part, and uses the same algorithms and options. Only the starting point, or the cost table entry, is varied.

System Requirements

For 3.1i, there is a new preferred method for setting up system requirements. The following two subsections describe each of these methods—the new and the old.

New Preferred Method

For the new method, you create a file, which can be sourced on remote systems, to set up the environment. This file is sourced with the Bourne Shell so it must be in the correct format. To source the file, follow these steps:

1. Create a file with a fixed path that can be accessed by all the systems you are using. For example:

   /net/$ {*nodename***} /home/jim/parmsetup**

2. Add the lines to set up the XILINX environment variable and the path.

   Example for HP systems:

   export XILINX=/net/${*nodename***} /home/jim/xilinx**
   export PATH=$XILINX/bin/hp: /usr/bin: /usr/sbin

   export SHLIB_PATH=$XILINX/bin/hp

   Example for SUN Solaris systems:

   export XILINX=/net/${*nodename***} /home/jim/xilinx**

   export PATH=$XILINX/bin/sol: /usr/bin: /usr/sbin

   export LD_LIBRARY_PATH=$XILINX/bin/sol

   For mixed sets of systems, you need a more sophisticated script that can set up the proper environment.

3. After setting up this file, set the environment variable PAR_M_SETUPFILE to the name of your file.

   Example for C shell:

   setenv PAR_M_SETUPFILE /net/${*nodename***} /home/jim/parm-setup**

   Example for Bourne or Korn shells:

export PAR_M_SETUPFILE=/net/${*nodename***} /home/jim/parm-setup;**

Old Method

The following list describes the system requirements for running the turns engine.

- **rsh** must be located through the path variable.
- The executables required on the systems defined in the nodes file are
  - /bin/sh
  - par (must be located through path variable).
- The Turns Engine logs onto a node and then invokes PAR. The environment variables on the node are read from the node's .cshrc file (or equivalent); they are not passed from the host to the node. Therefore, all the Xilinx environment variables below must be defined in the .cshrc file. If not, the PAR process on the node will not be able to find the software or the licenses.
  - XILINX (points at Xilinx directory structure — must be a path accessible to both the machine from which the Turns Engine is run and the node).
  - LD_LIBRARY_PATH (supports par path for shared libraries — must be a path accessible to both the machine from which the Turns Engine is run and the node).
  - path (contains $XILINX/bin/$PLATFORM, where $PLATFORM is one of the following: sun, sol, hp, or rs6000)

To determine if everything is set up correctly, you can run the rsh command to the nodes to be used. Type the following.

```
rsh node_name /bin/sh -c par
```

If you get the usage message back on your screen, everything is set correctly.

Turns Engine Environment Variables

The environment variables below are interpreted by the Turns Engine manager.

- PAR_AUTOMNTPT—Specifies the network automount point. The Turns Engine uses network path names to access files. For example, a local path name to a file may be designs/cpu.ncd, but the network path name may be /home/*machine_name*/ivan/designs/cpu.ncd or /net/*machine_name*/ivan/designs/cpu.ncd. The PAR_AUTOMNT environment variable should be set to the value of the network automount point. The automount points for the examples above are /home and /net. The default value for PAR_AUTOMNT is /net.

  The line below sets the automount point to /nfs. If the current working directory is /usr/*user_name*/*design_name* on node mynode, the command **cd /nfs/mynode/usr/***user_name*/*design_name* is generated before PAR runs on the machine.

  ```
  setenv PAR_AUTOMNTPT /nfs
  ```

  The setting below does not issue a **cd** command; you are required to enter full paths for all of the input and output file names.

  ```
  setenv PAR_AUTOMNTPT ""
  ```

  The setting below tells the system that paths on the local workstation are the same as paths on remote workstations. This can be the case if your network does not use an automounter and all of the mounts are standardized, or if you do use an automounter and all mount points are handled generically.

  ```
  setenv PAR_AUTOMNTPT "/"
  ```

- PAR_AUTOMNTTMPPT—Most networks use the /tmp_mnt temporary mount point. If your network uses a temporary mount point with a different name, like /t_mnt, then you must set the PAR_AUTOMNTTMPPT variable to the temporary mount point name. In the example above you would set PAR_AUTOMNTTMPPT to /t_mnt. The default value for PAR_AUTOMNTTMPPT is /tmp_mnt.

- PAR_M_DEBUG—Causes the Turns Engine to run in debug mode. If the Turns Engine is causing errors that are difficult to correct, you can run PAR in debug mode in the following way.

1. Set the PAR_M_DEBUG variable.

   setenv PAR_M_DEBUG 1

2. Create a node list file containing only a single entry (one node).

   This single entry is necessary because if the node list contains multiple entries, the debug information from all of the nodes is intermixed, and troubleshooting is difficult.

3. Run PAR with the –m (multi-tasking mode) option.

   In debug mode, all of the output from all commands generated by the PAR run is echoed to the screen. There are also additional checks performed in debug mode, and additional information supplied to aid in solving the problem.

- PAR_M_SETUPFILE—See the "New Preferred Method" section for a discussion of this variable.

Starting the Turns Engine From the Command Line

The following is the PAR command line syntax to run the Turns Engine.

```
par -m nodelist_file -n #_of_iterations -s
#_of_iterations_to_save mapped_desgin.ncd
output_directory.dir
```

-m *nodelist_file* specifies the nodelist file for the Turns Engine run.

-n *#_of_iterations* specifies the number of place and route passes.

-s *#_of_iterations_to_save* saves only the best -s results.

*mapped design***.ncd** is the input NCD file.

*output_directory***.dir** is the directory where the best results (–s option) are saved. Files include placed and routed NCD, summary timing reports (DLY), pinout files (PAD), and log files (PAR).

Debugging

With the Turns Engine you may receive messages from the login process. The problems are usually related to the network or to environment variables.

- Network Problem—You may not be able to logon to the machines listed in the nodelist file.

- Try to ping the nodes by running the following command.

  ping *machine_name*

  You should get a message that the machine is alive. The ping command should also be in your path (UNIX cmd: which ping).

- Try to logon to the nodes using the command rsh *machine_name*. You should be able to logon to the machine. If you cannot, make sure rsh is in your path (UNIX cmd: which rsh). If rsh is in your path, but you still cannot logon, contact your network administrator.

- Try to launch PAR on a node by entering the following command.

  rsh *machine_name* **/bin/sh -c par**.

  This is the same command that the Turns Engine uses to launch PAR. If this command is successful, everything is set up correctly for the *machine_name* node.

- Environment Problem—logon to the node with the problem by entering the following UNIX command

  rsh *machine name*

  Check the $XILINX, $LD_LIBRARY_PATH, and $PATH variables by entering the UNIX command **echo $** *variable_name*.

  If these variables are not set correctly, check to make sure these variables are defined in your .cshrc file.

  Note Some, but not all, errors in reading the .cshrc may prevent the rest of the file from being read. These errors may need to be corrected before the XILINX environment variables in the .cshrc are read. The error message /bin/sh: par not found indicates that the environment in the .cshrc file is not being correctly read by the node.

Screen Output

When PAR is running multiple jobs and is not in multi-tasking mode, output from PAR is displayed on the screen as the jobs run. When PAR is running multiple jobs in multi-tasking mode, you only see information regarding the current status of the Turns Engine. For example, when the job described in the "Turns Engine Overview" section is executed, the following screen output would be generated.

```
Starting job 5_5_1 on node jupiter
Starting job 5_5_2 on node mars
Starting job 5_5_3 on node mercury
Starting job 5_5_4 on node neptune
Starting job 5_5_5 on node pluto
```

When one of the jobs finishes, a message similar to the following displays.

```
Finished job 5_5_3 on node mercury
```

These messages continue until there are no jobs left to run, at which time "Finished" appears on your screen.

Note For HP workstations, you are not able to interrupt the job with Control-C as described below if you do not have Control-C set as the escape character. To set the escape character, refer to your HP manual.

You may interrupt the job at any time by pressing Control-C. If you interrupt the program, you see the following on your screen.

```
CONTRL-C interrupt detected.
Please choose one of the following options:
1.  Continue processing and ignore the interrupt.
2.  Normal program exit at next check point.
3.  Exit program immediately.
4.  Add a node for running jobs.
5.  Stop using a node.
6.  Display current status.

Enter choice  - - >
```

Choices are described below.

1. **Continue processing and ignore the interrupt**—self-explanatory.
2. **Normal program exit at next check point**—allows the Turns Engine to wait for all jobs to finish before terminating. PAR is

allowed to generate the master PAR output file (PAR), which describes the overall run results

When you select option 2, a secondary menu appears as shown below.

How would you like to handle the currently running job?
1. Allow jobs to finish.
2. Halt jobs at next checkpoint.
3. Halt jobs immediately.

Enter choice - - >

a) **Allow jobs to finish** — current jobs finish but no other jobs start if there are any. For example, if you are running 100 jobs (–n 100) and the current jobs running are 5_5_49 and 5_5_50, when these jobs finish, job 5_5_51 is not started.

b) **Halt jobs at next checkpoint** — all current jobs stop at the next checkpoint; no new jobs are started.

c) **Halt jobs immediately** — all current jobs stop immediately; no other jobs start

3. **Exit program immediately** — all running jobs stop immediately (without waiting for running jobs to terminate) and PAR exits the Turns Engine.

4. **Add a node for running jobs** — allows you to dynamically add a node on which you can run jobs. When you make this selection, you are prompted as follows.

   Input the name of the node to be added to the list

   After you enter the node name, a job starts immediately on that node and a “Starting job” message is displayed.

5. **Stop using a node** — allows you to remove a node from the list so that no job runs on that node.

   If you select **Stop using a node**, you must also select from the following options.

   Which node do you wish to stop using?
   1. jupiter
   2. mars
   3. mercury

   Enter number identifying the node.(<CR> to ignore)

Enter the number identifying the node. If you enter a legal number, you are asked to make a selection from this menu.

Do you wish to
1.Terminate the current job immediately and resubmit.
2.Allow the job to finish.
Enter number identifying choice. (<CR> to ignore)

The options are described below.

a) **Terminate the current job immediately and resubmit**—halts the job immediately and sets it up again to be run on the next available node. The halted node is not used again unless it is enabled by the "add" function.

b) **Allow the job to finish**—finishes the node's current job, then disables the node from running additional jobs.

Note The list of nodes described above is not necessarily numbered in a linear fashion. Nodes that are disabled are not displayed. For example, if NODE2 is disabled, the next time "Stop using a node" is opted, the following is displayed.

Which node do you wish to stop using?

1. jupiter
3. mercury

Enter number identifying the node. (<CR> to ignore)

6. **Display current status** — displays the current status of the Turns Engine. It shows the state of nodes and the respective jobs. Here is a sample of what you would see if you chose this option.

ID	NODE	STATUS	JOB	TIME
1.	jupiter	Job Running	5_5_10	02:30:45
2.	mars	Job Running	5_5_11	02:28:03
3.	mercury	Not Available		
4.	neptune	Pending Term	5_5_12	02:20:01
5.	pluto	Job Running	5_5_13	02:20:01
6.	venus	Idle		
7.	earth	Job Running	5_5_12	25

Each entry is described below:

- jupiter has been running job 5_5_10 for approximately 2 1/2 hours.
- mars has been running job 5_5_11 for approximately 2 1/2 hours.
- mercury has been deactivated by the user with the “Stop using a node” option or it was not an existing node or it was not running. Nodes are “pinged” to see if they exist and are running before attempting to start a job.
- neptune has been halted “immediately” with job resubmission. The Turns Engine is waiting for the job to terminate. Once this happens the status is changed to “not available”.
- pluto has been running job 5_5_13 for 2 hours 20 minutes.
- venus has finished its current job and is available for another. When you see the “Idle” message, it usually means that no other jobs are available.
- earth is running job 5_5_12. This job was resubmitted when neptune was dropped. It has been running for 25 seconds. It is unlikely that you will see the same job listed twice (as in the sample above) since the job pending termination usually finishes very quickly.

There is also a status named “Job Finishing”. This appears if the Turns Engine has been instructed to halt the job at the next checkpoint.

Command Line Examples

Following are a few examples of PAR command lines and a description of what each does.

Example 1:

The following command places and routes the design in the file input.ncd and writes the placed and routed design to output.ncd.

```
par input.ncd output.ncd
```

Example 2:

The following command skips the placement phase and preserves all routing information without locking it (re-entrant routing). Then it runs up to 999 passes of the router or stops upon completion and

conformance to timing constraints found in the pref.pcf file. Then it runs three delay-based cleanup router passes. If the design is already completely routed, the effect of this command is to just run three delay-based cleanup passes.

```
par  -k -i 999 -c 0 -d 3 input.ncd output.ncd pref.pcf
```

Example 3:

The following command runs 20 place and route iterations at overall effort level 3. The mapping of the overall level (–ol) to placer effort level (–pl) and router effort level (–rl) depends on the device to which the design was mapped, and placer level and router level do not necessarily have the same value. The iterations begin at cost table entry 5. Only the best 3 output design files are saved. The output design files (in NCD format) are placed into a directory called results.dir.

```
par  -n 20  -ol 3  -t 5  -s 3  input.ncd  results.dir
```

Now, if you wanted to run two passes of cost-based and delay-based cleanup on the three designs saved (without running placement), you would enter this command for each design.

```
par  -k -i 0 -c 2 -d 2  input.ncd  output.ncd
```

Example 4:

The following command copies the input design to the output design. The placement and routing phases are skipped completely. Since a delay file is generated as a result of the command, you can use these options to check the delay times in your design without having PAR change any of the design's placement or routing.

```
par  -pr  input.ncd  output.ncd
```

Example 5:

The following command allows re-entrant routing. Use this command when your design is only partially routed and you want to complete it or when the design does not meet your timing constraints and additional routing passes are needed to meet the constraints. Placement and placement optimization are skipped. In this case up to 30 router passes are run (you could run up to 2000). This may result in local rip-up and reroute if 20 router passes are run with no progress.

```
par -k -i 30 input.ncd output.ncd
```

Example 6:

The following command gives you a delay report for a placed and routed file without modifying the file.

```
par -pwr input.ncd input.ncd
```

Example 7:

The following command runs PAR (using the Turns Engine) on all nodes listed in the file named “allnodes”. It runs 10 place and route passes at placer effort level 3 and router effort level 2 on the file “mydesign.ncd”. It runs one cost-based cleanup pass of the router.

```
par -m allnodes -pl 3 –rl 2 -n 10 -i 10 -c l myde-
sign.ncd output.dir
```

Halting PAR

Note You are not able to halt PAR with Control-C as described below if you do not have Control-C set as the interrupt character. To set the interrupt character, enter **stty intr ^V^C** in the .login file or .cshrc file.

To halt a PAR operation, enter Control-C. In a few seconds, this message appears.

```
CNTRL-C interrupt detected.

Please choose one of the following options:
1. Continue processing and ignore the interrupt.
2. Normal program exit at next check point.
   This will result in saving the best results so
   far,
   after concluding current processing.
3. Exit program immediately.
4. Display Failing Timespec Summary.
5. Cancel the current job and move to the next one
   at
    the next check point.
Enter choice -->
```

If you have no failing time specifications or are not using the -n option, Options 4 and 5 display as follows.

```
  4. Display Failing Timespec Summary.
    (Not applicable: Data not available)
```

```
5. Cancel the current job and move to the next one
at
 the next check point.
 (Not applicable: Not a multi-run job.)
```

You then select one of the five options shown on the screen. The options work in this way.

- Option 1—this option causes PAR to continue operating as before the interruption. PAR then runs to completion.
- Option 2—this option continues the current place / route iteration until one of the following "check points".
  - ♦ After constructive placement
  - ♦ After the current optimization pass
  - ♦ After the current routing iteration

    The system then exits the PAR run and saves an intermediate output file containing the results up to the check point.

    If you use this option, you may continue the PAR operation at a later time. To do this, you must look in the PAR report file to find the point at which you interrupted the PAR run. You can then run PAR on the output NCD file produced by the interrupted run, setting command line options to continue the run from the point at which it was interrupted.

    Option 2 halt during routing may be helpful if you notice that the router is performing multiple passes without improvement, and it is obvious that the router will not achieve 100% completion. In this case, you may want to halt the operation before it ends and use the results to that point instead of waiting for PAR to end by itself.
- Option 3—this option stops the PAR run immediately. You do not get any output file for the current place / route iteration. You do, however, still have output files for previously completed place / route iterations.
- Option 4—PAR displays the contents of the ITR file on the screen and then resumes execution.
- Option 5—Terminates current iteration if you have used the -n option and continues the next iteration.

Note If you started the PAR operation from the Design Manager as a background process on a workstation, you must bring the process to the foreground using the fg command before you can halt the PAR operation.

After you run PAR, you can use the FPGA Editor on the NCD file to examine and edit the results. You can also perform a static timing analysis using TRACE or the Timing Analyzer. When the design is routed to your satisfaction, you can input the resulting NCD file into the Xilinx Development System's BitGen program. BitGen creates files that are used for downloading the design configuration to the target FPGA. For details on BitGen, see the "BitGen" chapter.

Chapter 13

PIN2UCF

This program is compatible with the following families.

- Spartan / XL / -II
- Virtex / -E / -II
- XC9500 / XL / XV
- XC4000E / L / EX / XL / XV / XLA
- XC3000A / L
- XC3100A / L
- XC5200

This chapter describes PIN2UCF. The chapter contains the following sections.

- "PIN2UCF"
- "PIN2UCF Syntax"
- "PIN2UCF Files"
- "PIN2UCF Options"
- "PIN2UCF Scenarios"

PIN2UCF

PIN2UCF is a program that generates pin locking constraints in a UCF file by reading a placed NCD file for FPGAs or GYD file for CPLDs. PIN2UCF writes its output to an existing UCF file. If there is no existing UCF file, PIN2UCF creates a new file. The following figure shows the flow through PIN2UCF.

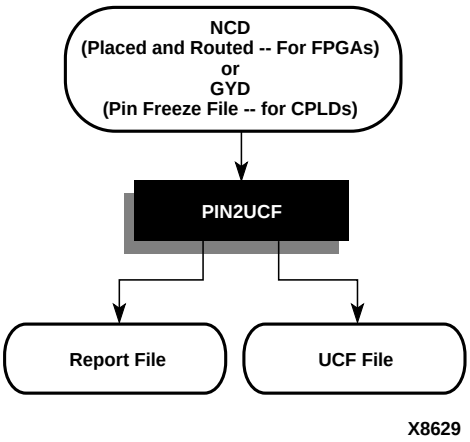


Figure 13-1 PIN2UCF Flow

The PIN2UCF is used to back-annotate pin locking constraints to the UCF file from a successfully placed and routed design (FPGAs) or successfully fitted design (CPLDs).

The program extracts pin locations and logical pad names from an existing NCD or GYD file and writes this information to a UCF file.

Pin locking constraints are written to a PINLOCK section in the UCF file. The PINLOCK section begins with the statement #PINLOCK BEGIN and ends with the statement #PINLOCK END. By default, PIN2UCF does not write conflicting constraints to a UCF file. Prior to creating a PINLOCK section, if PIN2UCF discovers conflicting constraints, it writes information to a report file, named pinlock.rpt.

The pinlock.rpt file has two sections: Constraint Conflicts Information and List of Errors and Warnings.

- The Constraints Conflicts Information section does not display if there are fatal input errors, for example, missing inputs or invalid inputs. However, the created report file contains the List of Errors and Warnings.
- The Constraints Conflicts Information section has two subsections.
  - Net name conflicts on the pins
  - Pin name conflicts on the nets

If there are no conflicting constraints, both subsections under the Constraint Conflicts Information section contain a single line indicating that there are no conflicts.

- The List of Errors and Warnings displays only if there are errors or warnings.

User-specified pin locking constraints are never overwritten in a UCF file. However, if the user-specified constraints are exact matches of PIN2UCF generated constraints, a pound sign (#) is added in front of all matching user-specified location constraint statements. The pound sign indicates that a statement is a comment. To restore the original UCF file (the file without the PINLOCK section), remove the PINLOCK section and delete the pound sign from each of the user-specified statements.

Note PIN2UCF does not check if existing constraints in the UCF file are valid pin locking constraints.

PIN2UCF writes to an existing UCF file under the following conditions.

- The contents in the PINLOCK section are all pin lock matches and there are no conflicts between the PINLOCK section and the rest of the UCF file.
- The PINLOCK section contents are all comments and there are no conflicts outside the PINLOCK section.
- There is no PINLOCK section and no other conflicts in the UCF file.
- Comments inside an existing PINLOCK section are never preserved by a new run of PIN2UCF.
- If PIN2UCF finds a CSTTRANS comment, it equates “INST *name*” to “NET name” and then checks for comments.

PIN2UCF Syntax

To invoke PIN2UCF from the UNIX or DOS command line, enter the following.

pin2ucf {*ncd_file*.**ncd** | *pin_freeze_file*.**gyd**} [**-r** *report_file_name* **-o** *output*.**ucf**]

ncd_file or *pin_freeze_file* must be the name of an existing file.

PIN2UCF Files

This section describes the PIN2UCF input and output files.

Input Files

Input to PIN2UCF can be either of the following files.

- NCD file—The minimal requirement is a placed NCD file, but you would normally use a placed and routed NCD file that meets (or is fairly close to meeting) timing specifications.
- GYD file—The PIN2UCF pin locking utility replaces the old GYD file mechanism that was used by CPLDs to lock pins. The GYD file is still available as an input guide file to control pin locking. Running PIN2UCF is the recommended method of pin locking to be used instead of specifying the GYD file as a Guide file.

Output Files

If there is no existing UCF file, PIN2UCF creates one. If a design.ucf file is not specified for PIN2UCF and a UCF file with the same root name exists in the same directory as the design file, the program appends to that file automatically unless there are constraint conflicts.

A pinlock.rpt file is written to the current directory by default. Use the –r option to write a report file to another directory. See the "–r (Write to a Report File)" section for more information.

PIN2UCF Options

The –o and –r options are the only PIN2UCF options.

–o (Output File Name)

-o *outfile*[**.ucf**]

The –o option specifies the name of the output UCF file for the design. The –o option is useful in the following ways.

- The UCF file used for the design has a different root name than the design name. By default, PIN2UCF writes a *ncd_file*.ucf file if –o is not specified. You can use this option to write the pin locking constraints to the UCF file with a different root name.
- You want to write a UCF file to a different directory.

–r (Write to a Report File)

-r *report_file_name*

The –r option writes the PIN2UCF report into the specified report file. If this option is not used, then a pinlock.rpt file is automatically written to the current directory.

PIN2UCF Scenarios

The following table describes the various PIN2UCF scenarios.

Scenarios	PIN2UCF Behavior	Files Created or Updated
No UCF file is present.	PIN2UCF creates a UCF file and writes the pin locking constraints to the UCF file.	pinlock.rpt *design_name*.ucf
UCF file is present. There are no pin locking constraints in the UCF file or this file contains some user-specified pin locking constraints outside of the PINLOCK section. None of the user specified constraints conflict with the PIN2UCF generated constraints.	PIN2UCF appends the pin locking constraints in the PINLOCK section to the end of the file.	pinlock.rpt *design_name*.ucf

Scenarios	PIN2UCF Behavior	Files Created or Updated
UCF file is present. This file contains some user-specified pin locking constraints either inside or outside of the PINLOCK section. Some of the user specified constraints conflict with the PIN2UCF generated constraints	PIN2UCF does not write the PINLOCK section. Instead it exits after providing an error message. It writes a list of conflicting constraints.	pinlock.rpt
UCF file is present. There are no pin locking constraints in the UCF file. There is a PINLOCK section in the UCF file generated from a previous run of PIN2UCF or manually created by the user. None of these constraints in the PINLOCK section conflict with PIN2UCF generated constraints	PIN2UCF writes a new PINLOCK section in the UCF file after deleting the existing PINLOCK section. The contents of the existing PINLOCK section are moved to the new PINLOCK section.	pinlock.rpt *design_name*.ucf

Chapter 14

TRACE

This program is compatible with the following families.

- XC3000A / L
- XC3100A / L
- XC4000E / L
- XC4000EX / XL / XV / XLA
- XC5200
- Spartan / XL / -II
- Virtex / -E / -II

This chapter describes TRACE®. The chapter contains the following sections.

- "TRACE"
- "TRACE Syntax"
- "TRACE Files"
- "TRACE Options"
- "Command Line Examples"
- "TRACE Input Details"
- "TRACE Output Details"
- "Halting TRACE"

TRACE

TRACE (Timing Reporter And Circuit Evaluator) provides static timing analysis of a design based on input timing constraints.

Note On the command line, the TRACE command is entered as **trce** (without an "A").

TRACE performs two major functions.

- Timing verification—the process of verifying that the design meets your timing constraints.
- Reporting—the process of enumerating input constraint violations and placing them into an accessible file. TRACE can be run on unplaced designs, completely placed and routed designs, or designs that are placed and routed to any degree of completion.

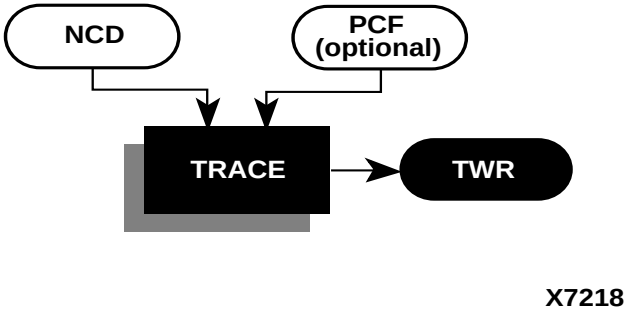


Figure 14-1 TRACE

TRACE Syntax

The following syntax runs TRACE.

trce [*options*] *design*[**.ncd**] [*constraint*[**.pcf**]]

Options can be any number of the TRACE options listed in the "TRACE Options" section. They do not need to be listed in any particular order. Separate multiple options with spaces.

Design[**.ncd**] is the name of the input physical design file. If you enter a file name with no extension, TRACE looks for an NCD file with the name you specified.

Constraint[**.pcf**] specifies the name of a timing physical constraints file. This file is used to define timing constraints for the design. If you do not specify a physical constraints file, TRACE looks for one with the same root name as the NCD file.

TRACE Files

This section describes the TRACE input and output files.

Input Files

Input files to TRACE are as follows.

- NCD file—a mapped design. The type of timing information you receive depends on whether the design is unplaced, placed only, or placed and routed.
- PCF file—an optional user-modifiable ASCII Physical Constraints File produced by MAP. The PCF file contains timing constraints used in the TRACE timing analysis.

Note The Viewlogic® CAE tools create a file with a .pcf extension when generating a plot of a Viewlogic schematic. This PCF file is not related to a Xilinx PCF file. Since TRACE automatically reads a PCF file with the same root name as your design file, make sure your directory does not contain a Viewlogic PCF file with the same root name as your NCD file.

Output Files

Output from TRACE is a timing report (TWR) file. There are three different types of timing reports: summary report, error report and verbose report. The type of report produced is determined by the

TRACE command line options you enter, as shown in the following table.

Table 14-1 TRACE Options and Reports

TRACE Option	Timing (TWR) Report Produced
No –e or –v	Summary report
-e	Error report
-v	Verbose report

Note In addition to the timing (TWR) report, you can specify -tsi on the command line to generate a Timespec Interaction Report (TSI). See the"TSI Report" section in this chapter for details.

TRACE Options

This section describes the options to the TRACE command.

–a (Advanced Analysis)

The –a option can only be used if you are not supplying any timing constraints (in a PCF file) to TRACE. The –a option writes out a timing report containing the following.

- An analysis that enumerates all clocks and the required OFFSETs for each clock.
- An analysis of paths having only combinatorial logic, ordered by delay.

This information is supplied in place of the default information for the output timing report type (summary, error, or verbose).

–e (Generate an Error Report)

-e [*limit*]

The –e option causes the timing report to be an error report instead of a summary report. See the "Error Report" section for a sample error report.

The report has the same root name as the input design and a .twr extension. You can assign a different root name for the report on the command line, but the extension must be .twr.

The *limit* is an integer limit on the number of items reported per constraint. The integer limit can be used to limit the number of items reported for each timing constraint in the report file (the default is 3 items).

–f (Execute Commands File)

-f *command_file*

The –f option executes the command line arguments in the specified *command_file*. For more information on the –f option, see the "–f Option" section of the "Introduction" chapter.

–l (Limit Timing Report)

The -l option is an integer limit on the number of items reported per constraint. The integer limit can be used to limit the number of items reported for each timing constraint in the report file (the default is 3 items).

–o (Output File Name)

The –o option specifies the name of the output timing report. The .twr extension is optional.

-o *outfile*[**.twr**]

The –o option specifies the name of the output timing report. The .twr extension is optional.

–s (Change Speed)

-s [*speed*]

The –s option overrides the device speed contained in the input NCD file and instead performs an analysis for the device speed you specify. The –s option applies to whichever report type you produce in this TRACE run. The option allows you to see if faster or slower speed grades meet your timing requirements.

The device *speed* can be entered with or without the leading dash. For example, both –s 3 and –s –3 are valid entries.

Some architectures support minimum timing analysis. The command line syntax for min timing analysis is: trace -s min. Do not place a leading dash before min.

Note The –s option only changes the speed grade for which the timing analysis is performed; it does not save the new speed grade to the NCD file.

–skew (Analyze Clock Skew for All Clocks)

-skew

This -skew option analyzes clock skew for all clocks including those using non-dedicated clock routing resources.

–stamp (Generates STAMP timing model files)

-stamp *stampfile design*.**ncd**

Note A *stampfile* entry is required before the NCD file entry for the 3.1i release.

When you specify the -stamp option, TRACE generates a pair of STAMP timing model files stampfile.mod and stampfile.data that characterize the design's timing.

The STAMP compiler can be used for any board when performing static timing analysis.

There are four methods of running TRACE with the STAMP option to obtain a complete STAMP model report.

- Run with advanced analysis (-a)
- Run using default analysis (that is, with no constraint file and without advanced analysis).
- Construct constraints to cover all paths in the design.
- Run using the unconstrained path report (-u option) for constraints which only partially cover the design.

For either of the last two options, you should not have any path controls or TIGs or be aware that those paths are not part of the model.

–tsi (Generate a Timespec Interaction Report)

-tsi *designfile*.**tsi** *designfile*.**ncd** *designfile*.**pcf**

When you specify the -tsi option, TRACE generates a Timespec Interaction Report. You must specify the design name on the command line. You can also specify the NCD and PCF files from which the Timespec Interaction Report analyzes constraints. If you do not specify the NCD and PCF files, TRACE looks for files that have the same root design name with .ncd and .pcf extensions.

–u (Report Uncovered Paths)

-u [*limit*]

The –u option reports delays for paths that are not covered by timing constraints. The option adds an "Unconstrained path analysis" constraint to your existing constraints. This constraint performs a default path enumeration on any paths for which no other constraints apply. The default path enumeration includes circuit paths to data and clock pins on sequential components and data pins on primary outputs.

The *limit* variable is an integer limit on the number of unconstrained paths reported for each timing constraint in the report file (the default is 3 items).

In the TRACE report, the following is included for the "Unconstrained path analysis" constraint.

- The minimum period for all of the uncovered paths to sequential components.

- The maximum delay for all of the uncovered paths containing only combinatorial logic.
- For a verbose report only, a listing of periods for sequential paths and delays for combinatorial paths. The list is ordered by delay in descending order, and the number of entries in the list can be controlled by specifying a limit when you enter the –v (Generate a Verbose Report) command line option.

–v (Generate a Verbose Report)

-v [*limit*]

The –v option generates a verbose report. The report has the same root name as the input design and a .twr extension. You can assign a different root name for the report on the command line, but the extension must be .twr.

The *limit* variable is an integer limit on the number of items reported per constraint. The integer limit can be used to limit the number of items reported for each timing constraint in the report file (the default is 3 items).

Command Line Examples

The following command verifies the timing characteristics of the design named design1.ncd, generating a summary timing report. Timing constraints contained in the file group1.pcf are the timing constraints for the design. This generates the report file design1.twr.

```
trce design1.ncd group1.pcf
```

The following command produces a file listing all delay characteristics for the design named design1.ncd, using the timing constraints contained in the file group1.pcf. The verbose report file is called output.twr.

```
trce –v design1.ncd group1.pcf –o output.twr
```

The following command analyzes the file design1.ncd and reports on the three worst errors for each constraint in timing.pcf. The report is called design1.twr.

```
trce   –e 3   design1.ncd   timing.pcf
```

The following command generates a TSI report in addition to a summary timing report. The TSI report is called design1.tsi (specified

on the command line). The summary timing report is called design1.twr.

```
trce –tsi design1.tsi design1.ncd timing.pcf
```

TRACE Input Details

Input to TRACE is a mapped NCD design and an optional physical constraints (PCF) file based upon timing constraints that you specify. Constraints can indicate such things as clock speed for input signals, the external timing relationship between two or more signals, absolute maximum delay on a design path, or a general timing requirement for a class of pins.

TRACE Output Details

TRACE output is an ASCII timing report file that enables you to see how well the timing constraints for the design have been met. The file is written into your current working directory and has a .twr extension. The default name for the file is the same root name as the NCD file. You can designate a different root name for the file, but it must have a .twr extension. The extension .twr is assumed if not specified.

The timing report lists statistics on the design, any detected timing errors, and a number of warning conditions.

Timing errors indicate absolute or relative timing constraint violations. These include the following.

- Path delay errors—where the path delay exceeds the maximum delay constraint for a path.
- Net delay errors—where a net connection delay exceeds the maximum delay constraint for the net.
- Offset errors—where either the delay offset between an external clock and its associated data-in pin is insufficient to meet the internal logic's timing requirements or the delay offset between an external clock and its associated data-out pin exceeds the external logic's timing requirements.
- Net skew errors—where skew between net connections exceeds the maximum skew constraint for the net.

Timing errors may require design modifications, running PAR, or both.

Warnings point out potential problems such as circuit cycles or a constraint that does not define any paths.

Three types of reports are available. You determine the report type by entering the appropriate option entry on the UNIX or DOS command line or by selecting the type of report from the Timing Analyzer (see the "TRACE Options" section). Each type of report is described in the "Reporting with TRACE" section.

Timing Verification with TRACE

TRACE checks the delays in the NCD design file against your timing constraints. If delays are exceeded, TRACE issues the appropriate timing error.

Net Delay Constraints

The delay for a constrained net is checked to ensure that the routedelay is less than or equal to the netdelayconstraint.

routedelay ≤ **netdelayconstraint**

routedelay is the signal delay between the driver pin and the load pin(s) on a net. This is an estimated delay if the design is placed but not routed.

Any nets showing delays that do not meet this condition generate timing errors in the timing report.

Net Skew Constraints

Signal skew on a net with multiple load pins is the difference between minimum and maximum load delays.

signalskew = (maxdelay - mindelay)

maxdelay is the maximum delay between the driver pin and a load pin.

mindelay is the minimum delay between the driver pin and a load pin.

For constrained nets in the PCF, skew is checked to ensure that the signalskew is less than or equal to the maxskewconstraint.

signalskew ≤ **maxskewconstraint**

If the skew is found to exceed the maximum skew constraint, the timing report shows a skew error.

Path Delay Constraints

The pathdelay equls the sum of logic (component) delay, route (wire) delay, and setup time (if any), minus clock skew (if any).

```
pathdelay = logicdelay + routedelay + setuptime -
clockskew
```

The delay for constrained paths is checked to ensure that the path-delay is less than or equal to the maxpathdelayconstraint.

`pathdelay` ≤ **`maxpathdelayconstraint`**.

logicdelay is the pin-to-pin delay through a component.

routedelay is the signal delay between component pins in a path. This is an estimated delay if the design is placed but not routed.

setuptime (for clocked paths only) is the time that data must be present on an input pin before the arrival of the triggering edge of a clock signal.

clockskew (for register-to-register clocked paths only) is the difference between the amount of time the clock signal takes to reach the destination register and the amount of time the clock signal takes to reach the source register. Clock skew is discussed in the following section.

Paths showing delays that do not meet this condition generate timing errors in the timing report.

Clock Skew and Setup Checking

Clock skew must be accounted for in register-to-register setup checks. For register-to-register paths, the data delay must reach the destination register within a single clock period for the destination register. The timing analysis software ensures that any clock skew between the source and destination registers is accounted for in this check.

Note In default mode, that is, without using the -skew option, only dedicated clock resource skew accounting is performed. With the -skew option, non-dedicated clock skew accounting is also performed.

A setup check performed on register-to-register paths checks the following condition.

```
Slack = constraint + Tsk - (Tpath + Tsu)
```

constraint is the required time interval for the path, either specified explicitly by you with a FROM TO constraint, or derived from a PERIOD constraint.

Tpath is the summation of component and connection delays along the path (including the Tcko delay from the source register).

Tsu (setup) is the setup requirement for the destination register.**Tsk (skew)** is the difference between the arrival time for the destination register and the source register.

Negative slack indicates that a setup error may occur, because the data from the source register does not set up at the target register for a subsequent clock edge.

In the following figure, the clock skew Tsk is the delay from the clock input (CLKIOB) to register D (TclkD) less the delay from the clock input (CLKIOB) to register S (TclkS). Negative skew relative to the destination reduces the amount of time available for the data path, and positive skew relative to the destination register is truncated to zero.

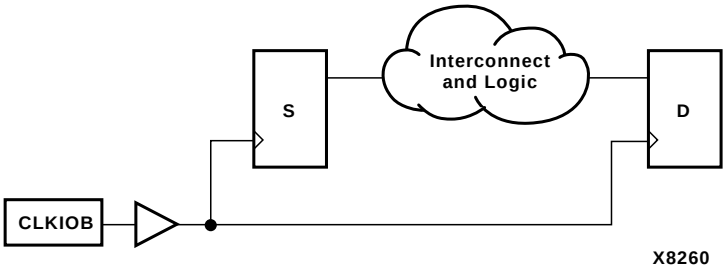


Figure 14-2 Clock Skew Example

Because the total clock path delay is used to determine the clock arrival times at the source register (TclkS) and the destination register (TclkD), this check still applies if the source and destination clocks originate at the same chip input but travel through different clock buffers and / or routing resources, as shown in the following figure.

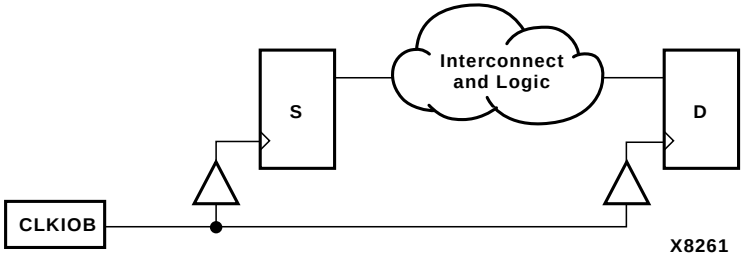


Figure 14-3 Clock Passing Through Multiple Buffers

When the source and destination clocks originate at different chip inputs, no obvious relationship between the two clock inputs exists for the timing software (because the software cannot determine the clock arrival time or phase information).

For FROM TO specifications, the software assumes you have taken into account the external timing relationship between the chip inputs. The software assumes both clock inputs arrive simultaneously, and the difference between the destination clock arrival time (TclkD) and the source clock arrival time (TclkS) does not account for any difference in the arrival times at the two chip clock inputs.

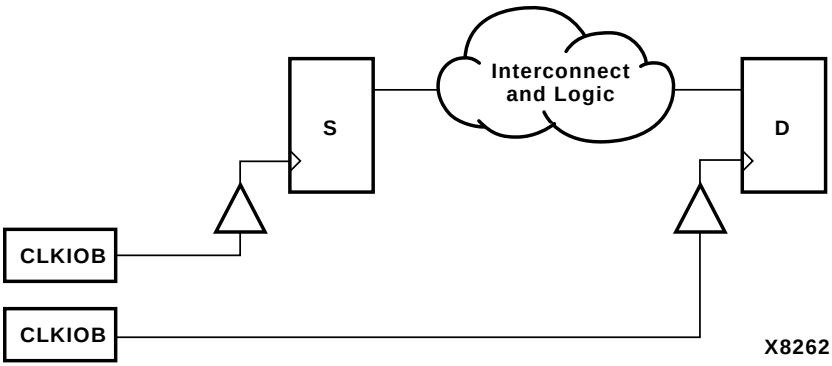


Figure 14-4 Clocks Originating at Different Device Inputs

The clock skew Tsk is not accounted for in setup checks covered by PERIOD constraints where the clock paths to the source and destination registers originate at different clock inputs.

Reporting with TRACE

The timing report produced by TRACE is an ASCII file prepared for a particular design. It reports statistics on the design, a summary of timing warnings and errors, and optional detailed net and path delay reports.

Note All TRACE reports are formatted for viewing in a monospace (non-proportional) font. If the text editor you use for viewing the reports uses a proportional font, the columns in the reports do not line up correctly.

This section covers the three different types of timing reports generated by TRACE. They are as follows.

- The summary report—Contains summary information, design statistics, and statistics for each constraint in the PCF file.

- The error report—Lists timing errors and associated net / path delay information.
- The verbose report—Lists delay information for all nets and paths.

In each type of report, the header specifies the command line used to generate the report, the type of report, the input design name, the optional input physical constraints file name, and device and speed data for the input NCD file. At the end of each report is a timing summary, which includes the following information.

- The number of timing errors found in the design. This information appears in all reports
- A timing score, showing the total amount of error (in picoseconds) for all timing constraints in the design
- The number of paths and nets covered by the constraints
- The number of route delays and the percentage of connections covered by timing constraints

Note The percentage of connections covered by timing constraints is given in a "% coverage" statistic. The statistic does not indicate the percentage of paths covered; it indicates the percentage of connections covered. Even if you have entered constraints that cover all paths in the design, this percentage may be less than 100%, since some connections are never included for static timing analysis (for example, connections to the STARTUP component).

- The number of nets covered by constraints
- A list of global statistics for the design

In the following sections, a description of each report is accompanied by a sample.

Following are some additional notes about timing reports.

- For any of the three types of reports, if you specify a physical constraints file that contains invalid data, a list of physical constraints file errors appears at the beginning of the report. These include errors in constraint syntax.

- In a timing report, a tilde (~) preceding a delay value indicates that the delay value is approximate. Values with the tilde cannot be calculated exactly because of excessive delays, resistance, or capacitance on the net, that is, the path is too complex too calculate accurately.

  The tilde (~) also means that the path may exceed the numerical value listed next to the tilde by as much as 20%. You can use the PENALIZE TILDE constraint to penalize these delays by a specified percentage (see the "Attributes, Constraints, and Carry Logic" chapter of the *Libraries Guide* for a description of the PENALIZE TILDE constraint).

- In a timing report, an "e" preceding a delay value indicates that the delay value is estimated because the path is not routed.

- In a timing report, an "R" appended to a delay value indicates the value was calculated for a rising signal, and an "F" indicates the value for a falling signal. An "X" indicates that the signal's edge is indeterminate. If rising and falling values are different, TRACE reports the appropriate delay.

- TRACE detects when a path cycles (that is, when the path passes through a driving output more than once), and reports the total number of cycles detected in the design. When TRACE detects a cycle, it disables the cycle from being analyzed. If the cycle itself is made up of many possible routes, each route is disabled for all paths which converge through the cycle in question and the total number is included in the reported cycle tally.

  A path is considered to cycle outside of the influence of other paths in the design. Thus if a valid path follows a cycle from another path, but actually converges at an input and not a driving output, the path is not disabled and will contain the elements of the cycle which may be disabled on another path.

- In Xilinx FPGAs,455 tristate buffer (TBUF) outputs are always routed on longlines. Pullup resistors may also be tied to these longlines. The timing effects of a TBUF / pullup combination is handled differently in the various FPGA architectures.

  - In XC3000A / L, XC3100A / L, 4000E / L, XC5200, and Spartan / XL designs, the delay associated with the longline is built into the component delay for the TBUF, and is not included in the delay reported for the net on the longline.

- In XC4000EX / XL / XV designs, the net delay on the longline is computed and reported as if the pullup (and not the TBUF output) is driving the net. If you want the delay to be computed with the TBUF driving the net, do not include any pullups at the output of the TBUF.

- Error counts reflect the number of path endpoints (register setup inputs, output pads) that fail to meet timing specifications, not the number of paths that fail the specification. Consider the following circuit.

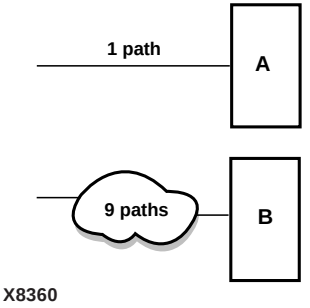


Figure 14-5 Error Reporting

If an error is generated at both the endpoints of A and B, the timing report would list two errors—one for each endpoint.

- On Virtex-II designs, the MAP program places information that is necessary to identify dedicated clocks in the PCF file. You must use a PCF generated by the MAP program to ensure timing analysis on these designs.

Data Sheet Reports

In 3.1i, the summary, error, and verbose reports contain a data sheet report. This report only includes IOs that are covered by the specified physical timing constraints, if any. A warning is issued if the report does not cover any IOs of the design due to the specified timing constraints. In the absence of a physical constraint file, all IO timing is analyzed and reported (less the effects of any default path tracing controls). Unconstrained path analysis can be used with a constraint file to increase the coverage of the report to include paths not explic-

itly specified in the constraints file. The report includes the source and destination PAD names, and either the propagation delay between the source and destination or the setup and hold requirements for the source relative to the destination. This report summarizes the following delay characteristics for the design:

- External setup / hold requirements

  The maximum setup and hold times of device data inputs are listed relative to each clock input. When two or more paths from a data input exist relative to a device clock input, the worst-case setup and hold times are reported. One worst-case setup and hold time is reported for each data input and clock input combination in the design.

  Following is an example of an external setup / hold requirement in the data sheet report:

```
Setup/Hold to clock ck1_i
---------------+------------+------------+
               | Setup to   |Hold to     |
Source Pad     |clk (edge)  |clk (edge)  |
---------------+------------+------------+
start_i        |2.816(R)    |0.000(R)    |
---------------+------------+------------+
```

- Clock-to-output propagation delays
- The maximum propagation delay from clock inputs to device data outputs are listed for each clock input. When two or more paths from a clock input to a data output exist, the worst-case propagation delay is reported. One worst-case propagation delay is reported for each data output and clock input combination.

  Following are two examples of clock-to-output propagation delays in the data sheet report:

```
Clock ck1_i to Pad
---------------+------------+
                |clk (edge)  |
Destination Pad|to PAD      |
```

```
---------------+------------+
out1_o         |   16.691(R)|
---------------+------------+
Clock to Setup on destination clock ck2_i
---------------+---------+---------+---------+--
|Src/Dest| Src/Dest| Src/Dest| Src/Dest|
Source Clock|Rise/Rise|Fall/Rise|Rise/Fall|Fall/
Fall|
---------------+---------+---------+---------+-+
ck2_i        | 12.647  |         |         |     |
ck1_i        |10.241   |         |         |     |
---------------+---------+---------+---------+-+
```

- Input-to-output propagation delays

  The maximum propagation delay from each device input to each device output is reported if a combinational path exists between the device input and output. When two or more paths exist between a device input and output, the worst-case propagation delay is reported. One worst-case propagation delay is reported for every input and output combination in the design.

  Following are examples of input-to-output propagation delays:

```
Pad to Pad
-----------------------------------------------
Source Pad   |Destination Pad|Delay  |
---------------+---------------+---------+
BSLOT0       |D0S            |37.534   |
BSLOT1       |D09            |37.876   |
BSLOT2       |D10            |34.627   |
BSLOT3       |D11            |37.214   |
CRESETN      |VCASN0         |51.846   |
```

```
CRESETN       |VCASN1        |51.846    |
CRESETN       |VCASN2        |49.776    |
CRESETN       |VCASN3        |52.408    |
CRESETN       |VCASN4        |52.314    |
CRESETN       |VCASN5        |52.314    |
CRESETN       |VCASN6        |51.357    |
CRESETN       |VCASN7        |52.527    |
---------------+---------------+---------
```

There are four methods of running TRACE to obtain a complete data sheet report.

- Run with advanced analysis (-a)
- Run using default analysis (that is, with no constraint file and without advanced analysis)
- Construct constraints to cover all paths in the design
- Run using the unconstrained path report for constraints which only partially cover the design

For either of the last two options, you should not have any path controls or TIGs or be aware that those paths will not be part of the report.

Guaranteed Setup and Hold Reporting

Guaranteed setup and hold values obtained from speed files are used in the data sheet reports for IOB input registers when these registers are clocked by specific clock routing resources and when the guaranteed setup and hold times are available for a specified device and speed.

Specific clock routing resources are clock networks that originate at a clock IOB, use a clock buffer to reach a clock routing resource and route directly to IOB registers.

Guaranteed setup and hold times are also used for reporting of input OFFSET constraints.

The following figure and text describes the external setup and hold time relationships.

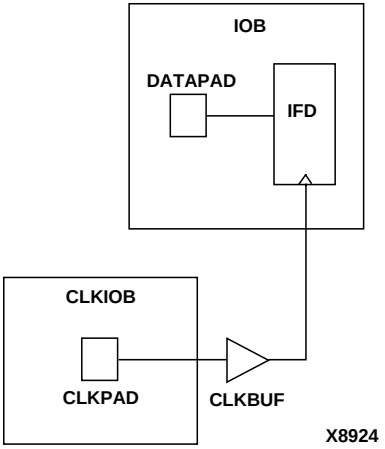


Figure 14-6 Guaranteed Setup and Hold

The pad CLKPAD of clock input component CLKIOB drives a global clock buffer CLKBUF, which in turn drives an input flip-flop IFD. The input flip-flop IFD clocks a data input driven from DATAPAD within the component IOB.

Setup Times

The external setup time is defined as the setup time of DATAPAD within IOB relative to CLKPAD within CLKIOB. When a guaranteed external setup time exists in the speed files for a particular DATAPAD and the CLKPAD pair and configuration, this number is used in timing reports. When no guaranteed external setup time exists in the speed files for a particular DATAPAD and CLKPAD pair, the external setup time is reported as the maximum path delay from DATAPAD to the IFD plus the maximum IFD setup time, less the minimum of maximum path delay(s) from the CLKPAD to the IFD.

Hold Times

The external hold time is defined as the hold time of DATAPAD within IOB relative to CLKPAD within CLKIOB. When a guaranteed external hold time exists in the speed files for a particular DATAPAD and the CLKPAD pair and configuration, this number is used in timing reports.

When no guaranteed external hold time exists in the speed files for a particular DATAPAD and CLKPAD pair, the external hold time is reported as the maximum path delay from CLKPAD to the IFD plus the maximum IFD hold time, less the minimum of maximum path delay(s) from the DATAPAD to the IFD.

Summary Report

The summary report includes the name of the design file being analyzed, the device speed and report level, followed by a statistical brief that includes the summary information (timing errors, etc. described above) and design statistics. The report also list statistics for each constraint in the PCF file, including the number of timing errors for each constraint.

A summary report is produced when you do not enter an –e (error report) or –v (verbose report) option on the TRACE command line.

Two sample summary reports are shown below. The first sample shows the results without having a physical constraints file. The second sample shows the results when a physical constraints file is specified.

If no physical constraints file exists or if there are no timing constraints in the PCF file, TRACE performs default path and net enumeration to provide timing analysis statistics. Default path enumeration includes all circuit paths to data and clock pins on sequential components and all data pins on primary outputs. Default net enumeration includes all nets.

Note The summary report is formatted for viewing in a monospace (non-proportional) font. If the text editor you use for viewing the report uses a proportional font, the columns in the report do not line up correctly.

Summary Report (Without a Physical Constraints File Specified)

The following sample summary report represents the output of this TRACE command.

```
trce -o summary.twr ramb16_s1.ncd
```

The name of the report is summary.twr. No preference file is specified on the command line, and the directory containing the file ram16_s1.ncd did not contain a PCF file called ramb16_s1.pcf.

```
--------------------------------------------------------------------
Xilinx TRACE, Version 3.1i
Copyright (c) 1995-2000 Xilinx, Inc.  All rights reserved.
Design file:              ramb16_s1.ncd
Device,speed:             xc2v250,-6
Report level:             summary report
--------------------------------------------------------------------

WARNING:Timing  -  No  timing  constraints  found,  doing  default
enumeration.
Asterisk (*) preceding a constraint indicates it was not met.
--------------------------------------------------------------------
  Constraint                     | Requested  | Actual   | Logic

                                 |            |          | Levels
--------------------------------------------------------------------
  Default period analysis        |            | 2.840ns    | 2
--------------------------------------------------------------------
  Default net enumeration        |            | 0.001ns    |
--------------------------------------------------------------------

All constraints were met.

Data Sheet report:
-----------------
All values displayed in nanoseconds (ns)

Setup/Hold to clock clk
---------------+------------+------------+
               |  Setup to  |  Hold to   |
Source Pad     | clk (edge) | clk (edge) |
```

```
---------------+------------+------------+
ad0            |    0.263(R)|    0.555(R)|
ad1            |    0.263(R)|    0.555(R)|
ad10           |    0.263(R)|    0.555(R)|
ad11           |    0.263(R)|    0.555(R)|
ad12           |    0.263(R)|    0.555(R)|
ad13           |    0.263(R)|    0.555(R)|
ad2            |    0.263(R)|    0.555(R)|
ad3            |    0.263(R)|    0.555(R)|
ad4            |    0.263(R)|    0.555(R)|
ad5            |    0.263(R)|    0.555(R)|
ad6            |    0.263(R)|    0.555(R)|
ad7            |    0.263(R)|    0.555(R)|
ad8            |    0.263(R)|    0.555(R)|
ad9            |    0.263(R)|    0.555(R)|
di             |    0.263(R)|    0.555(R)|
en             |    1.407(R)|    0.000(R)|
ssr            |    1.213(R)|    0.000(R)|
we             |    1.117(R)|    0.000(R)|
---------------+------------+------------+
Clock clk to Pad
---------------+------------+
               | clk (edge) |
Destination Pad|   to PAD   |
---------------+------------+
d0             |    7.496(R)|
---------------+------------+

Timing summary:
---------------
Timing errors: 0  Score: 0

Constraints cover 20 paths, 21 nets, and 21 connections (100.0%
coverage)

Design statistics:
   Minimum period:   2.840ns (Maximum frequency: 352.113MHz)
   Maximum combinational path delay:   6.063ns
   Maximum net delay:   0.001ns
Analysis completed Wed Mar  8 14:52:30 2000
--------------------------------------------------------------------
```

Summary Report (With a Physical Constraints File Specified)

The following sample summary report represents the output of this TRACE command

```
trce -o summary1.twr ramb16_s1.ncd clkperiod.pcf
```

The name of the report is summary1.twr. The timing analysis represented in the file were performed by referring to the constraints in the file clkperiod.pcf.

```
--------------------------------------------------------------------
Xilinx TRACE, Version 3.1i
Copyright (c) 1995-2000 Xilinx, Inc.  All rights reserved.

Design file:              ramb16_s1.ncd
Physical constraint file: clkperiod.pcf
Device,speed:             xc2v250,-6
Report level:             summary report
--------------------------------------------------------------------

Asterisk (*) preceding a constraint indicates it was not met.

--------------------------------------------------------------------
Constraint                          | Requested | Actual   | Logic
                                    |           |          | Levels
--------------------------------------------------------------------
TS01 = PERIOD TIMEGRP "clk" 10.0ns |          |         |
--------------------------------------------------------------------
OFFSET = IN 3.0 ns AFTER COMP "clk" TIMEG | 3.000ns| 8.593ns    2
RP "rams"
--------------------------------------------------------------------
* TS02 = MAXDELAY FROM TIMEGRP "rams" TO TI | 6.000ns  | 6.063ns   |2
  MEGRP "pads" 6.0 nS                     |          |          |
--------------------------------------------------------------------

1 constraint not met.

Data Sheet report:
-----------------
All values displayed in nanoseconds (ns)
```

```
Setup/Hold to clock clk
---------------+------------+------------+
               |  Setup to  |  Hold to   |
Source Pad     | clk (edge) | clk (edge) |
---------------+------------+------------+
ad0            |    0.263(R)|    0.555(R)|
ad1            |    0.263(R)|    0.555(R)|
ad10           |    0.263(R)|    0.555(R)|
ad11           |    0.263(R)|    0.555(R)|
ad12           |    0.263(R)|    0.555(R)|
ad13           |    0.263(R)|    0.555(R)|
ad2            |    0.263(R)|    0.555(R)|
ad3            |    0.263(R)|    0.555(R)|
ad4            |    0.263(R)|    0.555(R)|
ad5            |    0.263(R)|    0.555(R)|
ad6            |    0.263(R)|    0.555(R)|
ad7            |    0.263(R)|    0.555(R)|
ad8            |    0.263(R)|    0.555(R)|
ad9            |    0.263(R)|    0.555(R)|
di             |    0.263(R)|    0.555(R)|
en             |    1.407(R)|    0.000(R)|
ssr            |    1.213(R)|    0.000(R)|
we             |    1.117(R)|    0.000(R)|
---------------+------------+------------+
Clock clk to Pad
---------------+------------+
               | clk (edge) |
Destination Pad|   to PAD   |
---------------+------------+
d0             |    7.496(R)|
---------------+------------+

Timing summary:
---------------

Timing errors: 1  Score: 63

Constraints cover 19 paths, 0 nets, and 21 connections (100.0%
coverage)

Design statistics:
```

```
   Maximum path delay from/to any node:   6.063ns
   Maximum input arrival time after clock:   8.593ns

Analysis completed Wed Mar  8 14:54:31 2000
--------------------------------------------------------------------
```

When the physical constraints file includes timing constraints, the summary report lists the percentage of all design connections covered by timing constraints. If there are no timing constraints, the report shows 100 percent coverage. An asterisk precedes constraints that fail.

Error Report

The error report lists timing errors and associated net / path delay information. Errors are ordered by constraint and, within constraints, by slack (the difference between the constraint and the analyzed value, with a negative slack indicating an error condition). The maximum number of errors listed for each constraint is set by the limit you enter on the command line. The error report also contains a list of all time groups defined in the PCF file and all of the members defined within each group.

The main body of the error report lists all timing constraints as they appear in the input PCF file. If the constraint is met, the report simply states the number of items scored by TRACE, reports no timing errors detected, and issues a brief report line, indicating important information (for example, the maximum delay for the particular constraint). If the constraint is not met, it gives the number of items scored by TRACE, the number of errors encountered, and a detailed breakdown of the error.

For errors in which the path delays are broken down into individual net and component delays, the report lists each physical resource and the logical resource from which the physical resource was generated.

As in the other three types of reports, descriptive material appears at the top. A timing summary always appears at the end of the report. A sample error report follows.

Note The error report is formatted for viewing in a monospace (non-proportional) font. If the text editor you use for viewing the report uses a proportional font, the columns in the report do not line up correctly.

The following sample error report (error.twr) represents the output of this TRACE command.

trce -o error3.twr -e 3 ramb16_s1.ncd

```
--------------------------------------------------------------------
Xilinx TRACE, Version 3.1i
Copyright (c) 1995-2000 Xilinx, Inc.  All rights reserved.
Design file:              ramb16_s1.ncd
Physical constraint file: clkperiod.pcf
Device,speed:             xc2v250,-6
Report level:             error report
--------------------------------------------------------------------

====================================================================
Timing constraint: TS01 = PERIOD TIMEGRP "clk" 10.0ns ;
 0 items analyzed, 0 timing errors detected.
--------------------------------------------------------------------

====================================================================
Timing constraint: OFFSET = IN 3.0 ns AFTER COMP "clk" TIMEGRP
"rams" ;
 18 items analyzed, 0 timing errors detected.
 Maximum allowable offset is   8.593ns.
--------------------------------------------------------------------

====================================================================
Timing constraint: TS02 = MAXDELAY FROM TIMEGRP "rams" TO TIMEGRP
"pads" 6.0 nS  ;
 1 item analyzed, 1 timing error detected.
 Maximum delay is   6.063ns.
--------------------------------------------------------------------
Slack:    -0.063ns path RAMB16 to d0 relative to
           6.000ns delay constraint

Path RAMB16 to d0 contains 2 levels of logic:
Path starting from Comp: RAMB16.CLKA (from CLK)
To                  Delay type         Delay(ns)  Physical Resource
                                                      Logical Resource(s)
-------------------------------------------------  --------
RAMB16.DOA0          Tbcko                 2.543R  RAMB16
                                                     RAMB16.A
```

```
IOB.O1                net (fanout=1)     e  0.001R  N$41
IOB.PAD               Tioop                 3.519R  d0
                                                     I$22
                                                     d0
------------------------------------------------
Total (6.062ns logic, 0.001ns route)       6.063ns
      (100.0% logic, 0.0% route)
--------------------------------------------------------------------------

1 constraint not met.

Data Sheet report:
-----------------
All values displayed in nanoseconds (ns)

Setup/Hold to clock clk
---------------+------------+------------+
               |  Setup to  |  Hold to   |
Source Pad     | clk (edge) | clk (edge) |
---------------+------------+------------+
ad0            |    0.263(R)|    0.555(R)|
ad1            |    0.263(R)|    0.555(R)|
ad10           |    0.263(R)|    0.555(R)|
ad11           |    0.263(R)|    0.555(R)|
ad12           |    0.263(R)|    0.555(R)|
ad13           |    0.263(R)|    0.555(R)|
ad2            |    0.263(R)|    0.555(R)|
ad3            |    0.263(R)|    0.555(R)|
ad4            |    0.263(R)|    0.555(R)|
ad5            |    0.263(R)|    0.555(R)|
ad6            |    0.263(R)|    0.555(R)|
ad7            |    0.263(R)|    0.555(R)|
ad8            |    0.263(R)|    0.555(R)|
ad9            |    0.263(R)|    0.555(R)|
di             |    0.263(R)|    0.555(R)|
en             |    1.407(R)|    0.000(R)|
ssr            |    1.213(R)|    0.000(R)|
we             |    1.117(R)|    0.000(R)|
---------------+------------+------------+

Clock clk to Pad
```

```
---------------+------------+
               | clk (edge) |
Destination Pad|   to PAD   |
---------------+------------+
d0             |    7.496(R)|
---------------+------------+

Table of Timegroups:
--------------------
TimeGroup clk:
Signals:
 CLK

TimeGroup pads:
BELs:
 di    ad10  en    ad11  ad12  clk   ad13  ad0   ad1   ad2   ad3
ad4   ad5   ad6
we    ad7   ad8   ad9   d0    ssr

TimeGroup rams:
BELs:
 RAMB16.A
Timing summary:
---------------

Timing errors: 1  Score: 63

Constraints cover 19 paths, 0 nets, and 21 connections (100.0%
coverage)

Design statistics:
   Maximum path delay from/to any node:   6.063ns
   Maximum input arrival time after clock:   8.593ns

Analysis completed Wed Mar  8 14:55:32 2000
------------------------------------------------------------------
```

Verbose Report

The verbose report is similar to the error report, providing more details on delays for all constrained paths and nets in the design.

Entries are ordered by constraint and, within constraints, by slack. The maximum number of items listed for each constraint is set by the limit you enter on the command line.

Note The data sheet report and STAMP model display skew values on non-dedicated clock resources that do not display in the default period analysis of the normal verbose report. The data sheet report and STAMP model must include skew because skew affects the external timing model. To display skew values in the verbose report, use the -skew option

The verbose report also contains a list of all time groups defined in the PCF file, and all of the members defined within each group.

As in the other types of reports, descriptive material appears at the top.

The body of the verbose report enumerates each constraint as it appears in the input physical constraints file, the number of items scored by TRACE for that constraint, and the number of errors detected for the constraint. Each item is described, ordered by descending slack. A Report line for each item provides important information, such as the amount of delay on a net and by how much the constraint is met.

For path constraints, if there is an error, the report indicates the amount by which the constraint is exceeded. For errors in which the path delays are broken down into individual net and component delays, the report lists each physical resource and the logical resource from which the physical resource was generated.

If there are no errors, the report indicates that the constraint passed and by how much. Each logic and route delay is analyzed, totaled, and reported.

Note The verbose report is formatted for viewing in a monospace (non-proportional) font. If the text editor you use for viewing the report uses a proportional font, the columns in the report do not line up correctly.

The following sample verbose report (verbose.twr) represents the output of this TRACE command.

```
trce -o verbose1.twr -v 1 ramb16_sl.ncd
```

```
--------------------------------------------------------------------
Xilinx TRACE, Version HEAD (HEAD)
```

```
Copyright (c) 1995-2000 Xilinx, Inc.  All rights reserved.
Design file:              ramb16_s1.ncd
Physical constraint file: clkperiod.pcf
Device,speed:             xc2v250,-6
Report level:       verbose report, limited to 1 item per constraint
--------------------------------------------------------------------------------

================================================================================
Timing constraint: TS01 = PERIOD TIMEGRP "clk" 10.0ns ;
 0 items analyzed, 0 timing errors detected.
--------------------------------------------------------------------------------

================================================================================
Timing constraint: OFFSET = IN 3.0 ns AFTER COMP "clk" TIMEGRP
"rams" ;
 18 items analyzed, 0 timing errors detected.
 Maximum allowable offset is   8.593ns.
--------------------------------------------------------------------------------
Slack:     5.593ns path en to RAMB16 relative to
           1.433ns delay constraint clk to RAMB16 and
           7.000ns offset en to clk
Data path en to RAMB16 contains 2 levels of logic:
Path starting from Comp: IOB.PAD
To                  Delay type         Delay(ns)  Physical Resource
                                                  Logical Resource(s)
-------------------------------------------------  --------
IOB.I                Tiopi                 0.877R  en
                                                   en
                                                   I$31
RAMB16.ENA           net (fanout=1)     e  0.001R  N$1
RAMB16.CLKA          Tbeck                 1.962R  RAMB16
                                                   RAMB16.A
-------------------------------------------------
Total (2.839ns logic, 0.001ns route)       2.840ns (to CLK)
      (100.0% logic, 0.0% route)

Clock path clk to RAMB16 contains 2 levels of logic:
Path starting from Comp: IOB.PAD
To                  Delay type         Delay(ns)  Physical Resource
                                                  Logical Resource(s)
-------------------------------------------------  --------
```

```
IOB.I                Tiopi                 0.877R  clk
                                                   clk
                                                   clk/new_buffer
BUFGMUX.I0           net (fanout=1)     e  0.001R  clk/new_buffer
BUFGMUX.O            Tgi0o                 0.554R  I$9
                                                   I$9
RAMB16.CLKA          net (fanout=1)     e  0.001R  CLK
-------------------------------------------------
Total (1.431ns logic, 0.002ns route)       1.433ns
      (99.9% logic, 0.1% route)
--------------------------------------------------------------------------------

================================================================================
Timing constraint: TS02 = MAXDELAY FROM TIMEGRP "rams" TO TIMEGRP
"pads" 6.0 nS  ;
 1 item analyzed, 1 timing error detected.
 Maximum delay is   6.063ns.
--------------------------------------------------------------------------------
Slack:    -0.063ns path RAMB16 to d0 relative to
           6.000ns delay constraint

Path RAMB16 to d0 contains 2 levels of logic:
Path starting from Comp: RAMB16.CLKA (from CLK)
To                  Delay type         Delay(ns)  Physical Resource
                                                    Logical Resource(s)
-------------------------------------------------  --------
RAMB16.DOA0          Tbcko                 2.543R  RAMB16
                                                   RAMB16.A
IOB.O1               net (fanout=1)     e  0.001R  N$41
IOB.PAD              Tioop                 3.519R  d0
                                                   I$22
                                                   d0
-------------------------------------------------
Total (6.062ns logic, 0.001ns route)       6.063ns
      (100.0% logic, 0.0% route)

--------------------------------------------------------------------------------

1 constraint not met.

Data Sheet report:
```

```
-----------------
All values displayed in nanoseconds (ns)
Setup/Hold to clock clk
---------------+------------+------------+
               |  Setup to  |  Hold to   |
Source Pad     | clk (edge) | clk (edge) |
---------------+------------+------------+
ad0            |    0.263(R)|    0.555(R)|
ad1            |    0.263(R)|    0.555(R)|
ad10           |    0.263(R)|    0.555(R)|
ad11           |    0.263(R)|    0.555(R)|
ad12           |    0.263(R)|    0.555(R)|
ad13           |    0.263(R)|    0.555(R)|
ad2            |    0.263(R)|    0.555(R)|
ad3            |    0.263(R)|    0.555(R)|
ad4            |    0.263(R)|    0.555(R)|
ad5            |    0.263(R)|    0.555(R)|
ad6            |    0.263(R)|    0.555(R)|
ad7            |    0.263(R)|    0.555(R)|
ad8            |    0.263(R)|    0.555(R)|
ad9            |    0.263(R)|    0.555(R)|
di             |    0.263(R)|    0.555(R)|
en             |    1.407(R)|    0.000(R)|
ssr            |    1.213(R)|    0.000(R)|
we             |    1.117(R)|    0.000(R)|
---------------+------------+------------+
Clock clk to Pad
---------------+------------+
               | clk (edge) |
Destination Pad|   to PAD   |
---------------+------------+
d0             |    7.496(R)|
---------------+------------+

Table of Timegroups:
--------------------
TimeGroup clk:
Signals:
 CLK
TimeGroup pads:
```

```
BELs:
 di    ad10  en    ad11  ad12  clk   ad13  ad0   ad1   ad2   ad3
ad4   ad5   ad6    we    ad7   ad8   ad9   d0    ssr

TimeGroup rams:
BELs:
 RAMB16.A

Timing summary:
---------------
Timing errors: 1  Score: 63

Constraints cover 19 paths, 0 nets, and 21 connections (100.0%
coverage)

Design statistics:
   Maximum path delay from/to any node:   6.063ns
   Maximum input arrival time after clock:   8.593ns

Analysis completed Wed Mar  8 15:05:31 2000
--------------------------------------------------------------------
```

TSI Report

The TSI (TimeSpec Interaction) report describes interactions among timing constraints in your design. The report lists three categories of constraints: exclusive, duplicate, and extracted.

- **Exclusive** coverage occurs when a particular constraint is the *only* one that covers some paths from, to, or between the start and end points listed in the report.
- **Duplicate** coverage occurs when a constraint covers some paths from, to, or between the start and end points, but one or more other constraints cover the same paths as well.
- **Extracted** coverage occurs when some paths from, to, or between the start and end points that were supposed to be covered by a particular constraint are instead covered by a higher priority constraint.

The TSI report does not list the paths themselves; it lists the start points as sources and the end points as destinations. You might see

sources listed without destinations. For example a source can be listed in the exclusive constraint category without any destination. This means paths from that source are covered in the exclusive constraints category. However, the destination is covered by another category, duplicate or extracted. Similarly, you might see destinations listed without sources. This means paths to that destination are covered by one category of constraints, but the source is covered by another. Also note that not every source has a corresponding destination. Paths do not always exist between start and end points. The report lists start and end points between which there *may* be paths.

Two design examples with sample TSI reports follow. Design Example 1 illustrates how the timing tools determine extracted coverage. Design Example 2 illustrates how the timing tools determine duplicate coverage.

Design Example 1 (with Sample TSI Report)

This design example illustrates extracted coverage. The following figure contains two source registers, S1 and S2, two destination registers D1 and D2, and common intermediate logic.

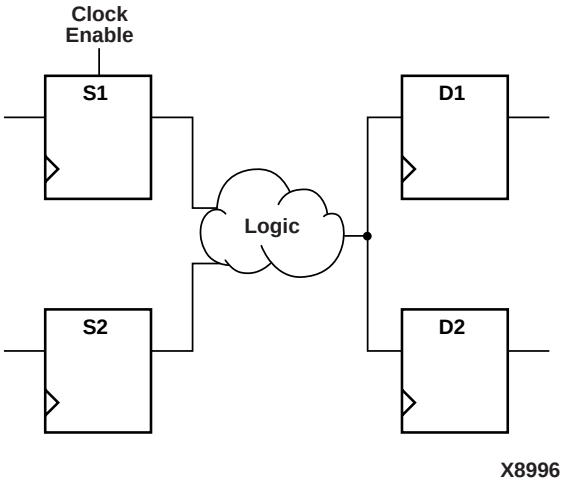


Figure 14-7 TSI Design Example 1

In this design, the registers S1, S2, D1 and D2 are all clocked by the same clock signal “CLOCK.” A combinatorial function in the logic is

the result of the two signals driven by registers S1 and S2. The four circuit paths in this example are as follows.

A={S1->D1}

B={S1->D2}

C={S2->D1}

D={S2->D2}

The following excerpt from the PCF file shows the related design constraints. A period constraint is defined for the clock signal, and a maximum delay constraint for the paths originating from the source register S1. (S1 has a clock enable signal.)

```
net “CLOCK” period=10

from BEL “S1” maxdelay=15
```

The period constraint defines the paths A, B, C and D in the design example. The maximum delay constraint defines the paths A and B. Since the maximum delay constraint takes priority over the period constraint, the paths A and B are covered by the maximum delay constraint only; they are *extracted* from period constraint coverage. The paths C and D *are* covered by the period constraint.

The following TSI report for this design represents the output of this TRACE command.

trce -tsi example1.tsi tsinew.ncd example1.pcf -o example1.twr

```
--------------------------------------------------------------------
Xilinx TRACE, Version 3.1i
Copyright (c) 1995-2000 Xilinx, Inc.  All rights reserved.
trce -tsi example1.tsi tsinew.ncd example1.pcf -o example1.twr
Design file: tsi.ncd
Physical constraint file: example1.pcf
Report level: timespec interaction report
--------------------------------------------------------------------
====================================================================
Timing constraint: from BEL "S1" maxdelay = 15
====================================================================
Paths from, to, or between the following sources and destinations
are covered exclusively by this timing constraint:
      Sources:
```

```
        S1
      Destinations:
The following constraints duplicately cover some or all paths from,
to, or between the sources and destinations listed below each
constraint:

  Constraint: net "CLOCK" period = 10
      Sources:
      Destinations:
        D1   D2
               The following constraints extracted some or all
                  paths from, to, or between the sources and
                  destinations listed below each constraint:
================================================================
Timing constraint: net "CLOCK" period = 10
================================================================
Paths from, to, or between the following sources and destinations
are covered exclusively by this timing constraint:
      Sources:
        S2
      Destinations:
The following constraints duplicately cover some or all paths from,
to, or between the sources and destinations listed below each
constraint:

  Constraint: from BEL "S1" maxdelay = 15
      Sources:
      Destinations:
        D1   D2

The following constraints extracted some or all paths from, to, or
between the sources and destinations listed below each constraint:

  Constraint: from BEL "S1" maxdelay = 15
      Sources:
        S1
      Destinations:
================================================================
------------------------------------------------
Pathtracing Controls: 12 entries (Default Settings)
------------------------------------------------
Standard Name: reg_sr_q         State: Disabled
```

```
        Trio   Trlat   Trri   Trpo
.
.
.
---------------------------
Active Pathtracing Controls
---------------------------
Disabled Signals (global):
    None
Disabled Pins (global):
    None
Disabled Delays (global):
        Trio   Trlat   Trri   Trpo
.
.
.
Constraint Disables:
                 None
-----------------------------------
Active Component Pathtracing Controls
-----------------------------------
Timespec interaction analysis completed Fri Feb 4 09:38:16 2000
-------------------------------------------------------------------
```

Design Example 2 (with Sample TSI Report)

This example illustrates duplicate coverage. It uses a different PCF file with the same design as the previous example. The design example follows.

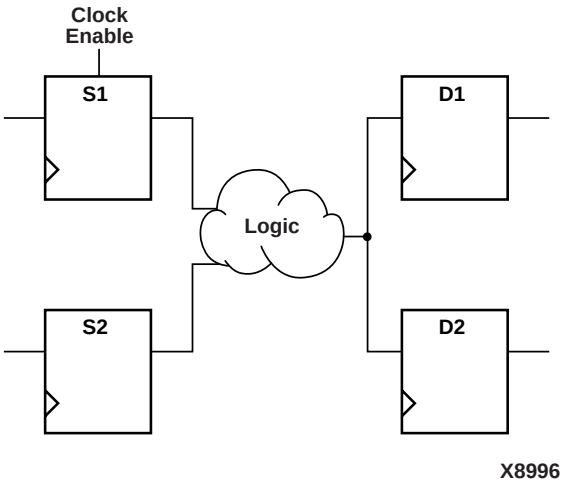


Figure 14-8 TSI Design Example 2

To recap the design, the registers S1, S2, D1 and D2 are all clocked by the same clock signal “CLOCK.” A combinatorial function in the logic is the result of the two signals driven by registers S1 and S2. The four circuit paths in the example are as follows.

A={S1->D1}

B={S1->D2}

C={S2->D1}

D={S2->D2}

In this example, the PCF file defines a maxdelay constraint for the paths from the source register S1 to the destination register D1.

```
net “CLOCK” period=10
from BEL “S1” to BEL “D1” maxdelay=15
```

The period constraint defines the paths A, B, C and D. The maxdelay constraint defines the set of paths A. The maxdelay constraint takes priority over the period constraint, but the timing tools are not path based and therefore cannot give precedence to the maxdelay constraint.

Note If precedence were given to the maxdelay constraint, the source S1 and destination D1 would have to be removed from coverage by the period constraint. This would also entail removing the paths defined by B and C. Since this is not the purpose of the constraint, the timing tools do not extract the paths affected by the period constraint.

In this situation, the path A is covered *duplicately* by both the period and the maxdelay constraints. The paths B, C, and D are covered by the period constraint alone.

Duplicate coverage may indicate that you need to refine your constraints. With reference to this example, the questions the designer might ask are:

- If the Clock Enable is used only on the S1 register, why doesn't the maxdelay constraint apply to all paths from S1 (as opposed to paths to D1)?
- If the S1 to D1 path is truly an exception to the period constraint, why isn't the S2 to D1 path also an exception?

To refine these constraints, the designer can consider the following:

- Define a clock period constraint using a timegrp (S1, D1). Include only S1, D1 and a slow period in the timegrp period constraint.
- Define another timegrp (S2, D2) period constraint listing only S2, D2 and a fast period.
- Define a constraint with any cross-group timing requirement.

The following TSI report for this design represents the output of this TRACE command.

```
trce -tsi example2.tsi tsinew.ncd example2.pcf -
o example2.twr
```

```
--------------------------------------------------------------------

Xilinx TRACE, Version 3.1i
Copyright (c) 1995-2000 Xilinx, Inc.  All rights reserved.
Design file:  tsi.ncd
Physical constraint file: example2.pcf
Report level:  timespec interaction report
--------------------------------------------------------------------

====================================================================
```

```
Timing constraint: from BEL "BEL_SOURCE.FFY" to BEL
"BEL_DEST.FFY"
maxdelay = 15
================================================================
Paths from, to, or between the following sources and destinations
are covered exclusively by this timing constraint:
Sources:
Destinations:

The following constraints duplicately cover some or all paths from,
to, or between the sources and destinations listed below each
constraint:
Constraint: net "CLOCK" period = 10
    Sources:
      BEL_SOURCE.FFY
    Destinations:
      BEL_DEST.FFY

The following constraints extracted some or all paths from, to, or
between the sources and destinations listed below each constraint:

================================================================
Timing constraint: net "CLOCK" period = 10
================================================================
Paths from, to, or between the following sources and destinations
are covered exclusively by this timing constraint:
    Sources:
      BEL_SOURCE.FFX
    Destinations:
      BEL_DEST.FFX

The following constraints duplicately cover some or all paths from,
to, or between the sources and destinations listed below each
constraint:

  Constraint: from BEL "BEL_SOURCE.FFY" to BEL "BEL_DEST.FFY"
maxdelay =15
    Sources:
      BEL_SOURCE.FFY
    Destinations:
      BEL_DEST.FFY
```

```
The following constraints extracted some or all paths from, to, or
between the sources and destinations listed below each constraint:
=================================================================

-------------------------------------------------
Pathtracing Controls: 12 entries (Default Settings)
-------------------------------------------------
Standard Name: reg_sr_q   State: Disabled
      Trio   Trlat   Trri   TrpoStandard Name: reg_sr_clk    State:
Disabled
.
.
.
---------------------------
Active Pathtracing Controls
---------------------------
Disabled Signals (global):
    None
Disabled Pins (global):
    None
Disabled Delays (global):
      Trio   Trlat   Trri  Trpo
      Topsc+Tpslic   Tops+Tpdsli   Topsc+Tpdslic
.
.
.
Constraint Disables:
   None

-----------------------------------
Active Component Pathtracing Controls
-----------------------------------

Timespec interaction analysis completed Wed Feb  2 15:51:55 2000
-----------------------------------------------------------------
```

Halting TRACE

To halt TRACE, enter CONTROL-C (on a workstation) or CONTROL-BREAK (on a PC). On a workstation, make sure that when you enter CONTROL-C, the active window is the window from which you invoked TRACE. The program prompts you to

confirm the interrupt. Some files may be left when TRACE is halted (for example, a TRACE report file or a physical constraints file), but these files may be discarded because they represent an incomplete operation.

Chapter 15

SPEEDPRINT

Speedprint is compatible with the following families.

- Spartan / XL / -II
- Virtex
- XC4000 / E / L / EX / XL / XV / XLA
- XC3000A / L
- XC3100A / L
- XC5200

This chapter contains the following sections.

- "SPEEDPRINT"
- "SPEEDPRINT Syntax"
- "SPEEDPRINT Options"
- "Example Commands" section
- "Example Outputs"

SPEEDPRINT

SPEEDPRINT lists important block delays for a specific device's speed grade in a convenient tabular format of an ASCII file. The program is not intended to replace data sheets, but rather should be used as a supplement. The SPEEDPRINT program reports devices and speed grades installed on your system.

For detailed information, see the *The Programmable Logic Data Book* or the data sheets at the Xilinx web site.

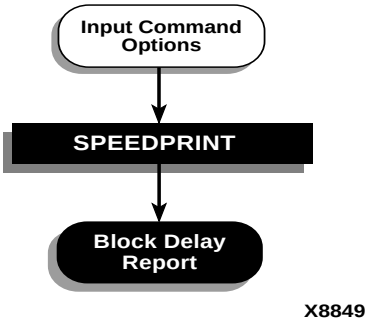


Figure 15-1 SPEEDPRINT

SPEEDPRINT Syntax

The following syntax runs speedprint.

speedprint [*options*] *device_name*

options can be any number of the SPEEDPRINT options listed in any order. Make sure to separate multiple options with spaces. See the next section for a description of these options.

To display a description of the SPEEDPRINT command and its options, enter **speedprint** or **speedprint -h**.

SPEEDPRINT Options

This section describes the options to the SPEEDPRINT command.

–min (Display Minimum Speed Data)

-min

The –m option displays minimum speed data for a device. This option overrides the –s option if both are used.

–s (Speed Grade)

-s [*speed_grade*]

The –s option with a speed_grade argument (for example, -4) displays data for the specified speed grade. If the –s option is

omitted, delay data for the default, which is the fastest speed grade, is displayed.

–t (Specify Temperature)

-t *temperature*

The –t option specifies the operating die temperature in degrees Celsius. If this option is omitted, the worst-case temperature is used.

–v (Specify Voltage)

-v *voltage*

The –v option specifies the operating voltage of the device in volts. If this option is omitted, the worst-case voltage is used.

Example Commands

The following table describes some example commands

Command	Description
speedprint	Prints usage message
speedprint xc4044xl	Uses the default speed grade
speedprint –s -3 xc4044xl speedprint -s 3 xc4044xl	Both displays block delays for speed grade -3
speedprint –v 3.0 -t 40 xc4044xl	Uses default speed grade at 3.0 volts and 40 degrees C
speedprint –s min xc4044xl speedprint -min xc4044xl	Both display data for the minimum speed grade.

Example Outputs

Following is the first part of a speed grade report for the XC4044XL device. The following command generates the displayed report.

speedprint xc4044xl

The default speed grade, temperature, and voltage settings are described at the beginning of the file.

```
Block delay report for a: x4044xl
```

```
Speed grade is: -09
Version id for speed file is: x1_0.45 1.24 PRELIMINARY xilinx
This speed file supports voltage adjustments over the range of
3.000000 to 3.600000 volts.
Temperature adjustments are supported over the junction
temperature range of 0.000000 to 85.000000 degrees Celsius
This report prepared for default temperature and voltage.
Note - this report is intended to present the effect of different
speedgrades and voltage/temperature adjustments on block delays,
for specific situations use the timing analyzer report instead.
Delays are reported in picoseconds, where a range of delays is
given they represent the fastest and slowest paths reported under
that name.
When a block is placed in a site normally used for another type
of block, a IOB placed in a Clock IOB site for example, small
variations in delay may occur which are not included in this report.
```

External Setup and Hold requirements for global clocks

Tpfhen	**4300**	**Tpfhep**	**0**	**Tpfsen**	**800**
Tpfsep	**10200-10800**	**Tphd**	**0**	**Tphed**	**0**
TPhen	4300	Tphep	0	Tphn	3800
Tphp	0	Tpsd	6800	Tpsed	9100
Tpsen	800	Tpsep	10200-10800	Tpsn	800
Tpsp	8600				

Delays for a CLB

Tah	2000	Tahds	0	Tahs	0
Taht	2000	Tahts	0	Tas	1959-2430
Tasc+Tas	3760	Tasc+Tass	3460	Tasc+Tast	3760
Tasc+Tasts	3460	Tasc+Tick	2670	Tasc+Tihck	3440
Tasc+Tiho	3760	Tasc+Tihto	4410	Tasc+Tilo	2989
Tasc+Tito	3639	Tascy	1800	Tasds	1660-2130
Tass	1660-2130	Tast	1950-2430	Tasts	1650-2130
Tbyp	140	Tcbyp	929	Tckd	0
Tckdi	0	Tckdt	0	Tckec	0

Tckhh0	0	Tckhh1	0	Tckhh2	0
Tcki	0	Tckih	0	Tckk	-7080-5410
Tcko	1470	Tckr	0	Tcto	1680
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.

`Delays for a IOB`

Tchio	2300	Tckpi	0	Tckpic	0
Tckpid	0	Tckpidc	0	Tckpim	0
Tckpimc	0	Tckpis	0	Tckpisc	0
Tckpisd	0	Tckpisdc	0	Tckps	0
Tckpsc	0	Tckpsd	0	Tckpsdc	0
Tecik	60	Tecok	-190	Tikec	0
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
Trpo	18050	Trri	18390	Ttshz	4100
Ttshz	4100	Ttsonf	3300	Ttsonfc	4150
Ttsons	5000	Ttsonsc	5850		

`Delays for a TBUF`

Tio	470	Toff	600	Ton	800

Chapter 16

BitGen

This program is compatible with the following families.

- XC3000A / L
- XC3100A / L
- XC4000E / L
- XC4000EX / XL / XV / XLA
- XC5200
- Spartan / XL / -II
- Virtex / -E / -II

This chapter describes BitGen. The chapter contains the following sections.

- "BitGen"
- "BitGen Syntax"
- "BitGen Files"
- "BitGen Options"

BitGen

BitGen produces a bitstream for Xilinx device configuration. After the design has been completely routed, it is necessary to configure the device so that it can execute the desired function. This is done with BitGen, Xilinx's bitstream generation program. BitGen takes a fully routed NCD (Circuit Description) file as its input and produces a configuration bitstream—a binary file with a .bit extension.

The BIT file contains all of the configuration information from the NCD file defining the internal logic and interconnections of the

FPGA, plus device-specific information from other files associated with the target device. The binary data in the BIT file can then be downloaded into the FPGA's memory cells or it can be used to create a PROM file (see the "PROMGen" chapter).

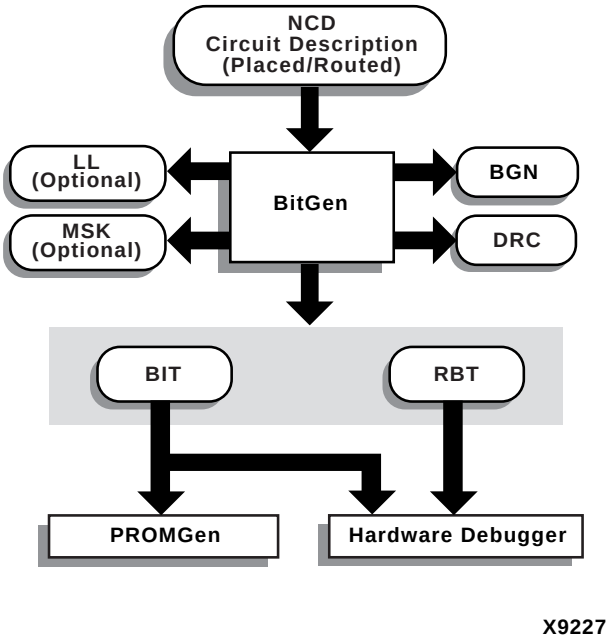


Figure 16-1 BitGen

BitGen Syntax

The following syntax creates a bitstream from your NCD file.

bitgen [*options*] *infile*[**.ncd**] [*outfile*] [*pcf_file*]

options is one or more of the options listed in the "BitGen Options" section.

Infile is the name of the NCD design for which you want to create the bitstream. You may specify only one design file, and it must be the first file specified on the command line.

You do not have to use an extension. If you do not, .ncd is assumed. If you do use an extension, it must be .ncd.

Outfile is the name of the output file. If you do not specify an output file name, BitGen creates one in the input file's directory. If you specify -l on the command line, the extension is .ll (see –l command line option). If you specify –m (see –m command line option), the extension is .msk. If you specify –b, the extension is .rbt. Otherwise the extension is .bit. If you do not specify an extension, BitGen appends one according to the aforementioned rules. If you do include an extension, it must also conform to the rules.

Pcf_file is the name of a physical constraints (PCF) file. BitGen uses this file to determine which nets in the design are critical for tiedown (see the "–t (Tie Unused Interconnect)" section). BitGen automatically reads the .pcf file by default. If the physical constraints file is the second file specified on the command line, it must have a .pcf extension. If it is the third file specified, the extension is optional; .pcf is assumed. If a .pcf file name is specified, it must exist, otherwise the input design name with a .pcf extension is read if that file exists.

A report file containing all of BitGen's output is automatically created under the same directory as the output file. The report file has the same root name as the output file and a .bgn extension.

BitGen Files

The input files that BitGen requires and the output files that BitGen generates are described below.

Input Files

Input to BitGen consists of the following files.

- NCD file—a physical description of the design mapped, placed and routed in the target device. The NCD file must be fully routed.
- PCF—an optional user-modifiable ASCII Physical Constraints File. If you specify a PCF file on the BitGen command line, BitGen uses this file to determine which nets in the design are critical for tiedown (see the "–t (Tie Unused Interconnect)" section).

Output Files

Output from BitGen consists of the following files.

- BIT file—a binary file with a .bit extension. The BIT file contains all of the configuration information from the NCD file defining the internal logic and interconnections of the FPGA, plus device-specific information from other files associated with the target device. The binary data in the BIT file can then be downloaded into the FPGA's memory cells or it can be used to create a PROM file (see the "PROMGen" chapter).
- RBT file—an optional "rawbits" file with an .rbt extension. The rawbits file consists of ASCII ones and zeros representing the data in the bitstream file. If you enter a –b option on the BitGen command line. an RBT file is produced in addition to the binary BIT file (see the "–b (Create Rawbits File)" section).
- LL file—an optional ASCII logic allocation file with an .ll extension. The logic allocation file indicates the bitstream position of latches, flip-flops, and IOB inputs and outputs. An LL file is produced if you enter a –l option on the BitGen command line (see the "–l (Create a Logic Allocation File)" section).
- MSK file—an optional mask file with an .msk extension. This file is used to compare relevant bit locations for executing a readback of configuration data contained in an operating FPGA. A MSK file is produced if you enter a –m option on the BitGen command line (see the "–m (Generate a Mask File)" section).
- BGN file—a report file containing information about the BitGen run.
- DRC file—a Design Rule Check (DRC) file for the design. A DRC runs and the DRC file is produced unless you enter a –d option on the BitGen command line (see the "–d (Do Not Run DRC)" section).

BitGen Options

Following is a description of the command line options and how they affect the behavior of BitGen.

Note For a complete description of the Xilinx Development System command line syntax, see the “Command Line” section of the “Introduction” chapter.

Options for the BitGen command are as follows.

–a (Tie All Interconnect)

Used with the -t option to force tiedown to fail if all nodes are not tied. This option also allows tiedown to implement user signals.

–b (Create Rawbits File)

Create a "rawbits" (*file_name*.rbt) file. The rawbits file consists of ASCII ones and zeros representing the data in the bitstream file.

If you are using a microprocessor to configure a single FPGA, you can include the rawbits file in the source code as a text file to represent the configuration data. The sequence of characters in the rawbits file is the same as the sequence of bits written into the FPGA.

–d (Do Not Run DRC)

Do not run DRC (Design Rule Check). Without the –d option, BitGen runs a DRC and saves the DRC results in two output files: the BitGen report file. (*file_name*.bgn) and the DRC file (*file_name*.drc). If you enter the –d option, no DRC information appears in the report file and no DRC file is produced.

Running DRC before a bitstream is produced detects any errors that could cause the FPGA to malfunction. If DRC does not detect any errors, BitGen produces a bitstream file (unless you use the –j option described in the “–j (No BIT File)” section).

You cannot disable the DRC with the –d option if you have specified a –t (Tie Unused Interconnect) option. The DRC always runs if you specify –t.

–f (Execute Commands File)

-f *command_file*

The –f option executes the command line arguments in the specified *command_file*. For more information on the –f option, see the "–f Option" section of the "Introduction" chapter.

–g (Set Configuration)

The –g option specifies the startup timing and other bitstream options for Xilinx FPGAs. The settings for the –g option depend on the design's architecture. These settings are described in the following sections.

- "–g (Set Configuration—XC3X00 Devices)"
- "–g (Set Configuration—XC4000 and Spartan)"
- "–g (Set Configuration—XC5200 Devices)"
- "–g (Set Configuration—Virtex/-E/-II and Spartan-II Devices)"

–g (Set Configuration—XC3X00 Devices)

The –g option has sub-options that represent settings you use to set the configuration for an XC3X00A/L design. These options have the following syntax.

bitgen -g *option*:*setting*

For example, to set the input signal thresholds to CMOS level instead of TTL level, use the following syntax.

```
bitgen -g inputs:CMOS
```

The following sections describe the startup sequences for the –g option applied to an XC3X00 design.

DonePin

Enables or disables internal pull-up on the DONE/$\overline{\text{PROGRAM}}$ (D/$\overline{\text{P}}$) pin. The Pullnone setting indicates there is no connection to the pull-up.

Use this option only if you are planning to connect an external pull-up resistor to this pin. The internal pull-up resistor has a value of 2 to 8 kohm and is automatically connected if you do not use this option.

The D/$\overline{P}$ pins configure an open-drain driver that requires a pull-up resistor to indicate the end of the configuration.

Architectures:	XC3000A/L, XC3100A/L
Settings:	Pullup, Pullnone
Default:	Pullup

DoneTime

Releases the DONE/$\overline{PROGRAM}$ (D/$\overline{P}$) pin one Cclk cycle before the IOBs become active (Before setting) or one Cclk cycle after the IOBs become active (After setting).

The After setting clearly indicates the end of the configuration process. The Before setting can be used to de-activate external configuration drivers so that they do not contend with active outputs on the same pin. The use of After would create a 1-Cclk-period contention. The alternative, using the LDC output, might cause a short contention spike. Before avoids these problems.

Architectures:	XC3000A/L, XC3100A/L
Settings:	Before, After
Default:	Before

Input

This option sets the FPGA design input-signal thresholds to TTL or CMOS level for interface capability. CMOS improves noise immunity and reduces static power consumption.

The special-purpose clock inputs, TCLKIN, BCLKIN, and $\overline{PWRDN}$ always require CMOS-level signals, even if the FPGA design input thresholds are specified as TTL compatible

Architectures:	XC3000A/L, XC3100A/L
Settings:	TTL, CMOS
Default:	TTL

LC_Alignment

Determines how length count is calculated to control when the device changes from configuration to user operation. The two methods of calculating length count, DONE Alignment and Length Count Alignment, are discussed in *The Programmable Logic Data Book*. The *FPGA Configuration Guidelines Application Note* also contains length count information.

Architectures:	XC3000A / L, XC3100A / L
Settings:	Length, DONE
Default	Length

Oscillator

This option specifies crystal oscillator options for XC3X00 series devices. The crystal oscillator is associated with the auxiliary clock buffer in the lower-right corner of the die.

The Disable option disables the FPGA crystal oscillator; Enable enables it. The EnableDiv2 option enables the oscillator and divides the crystal output frequency by two in order to guarantee a symmetrical clock signal

Architectures:	XC3000A / L, XC3100A / L
Settings:	Disable, Enable, EnableDiv2
Default	Disable

ReadBack

This option specifies readback options for XC3X00 families. After the FPGA design has been configured, the FPGA configuration data can be read back and compared with the original configuration data. Readback is initiated by a Low-to-High transition on the M0 / RTRIG pin. Once you give the readback command, external logic must drive the Cclk input to read back each data bit. The readback data appears on the $\overline{\text{RDATA}}$ pin.

The Disable option disables readback. The Once option enables a one-time readback and Command enables readback on command.

The Disable and Once options are used for design security. The Once option allows only one readback, typically performed during manufacturing. After this, readback can never be invoked again. If the FPGA device is powered by a standby battery and the

configuration source is removed, the FPGA design configuration data is completely secure from being read or copied.

Architectures:	XC3000A / L, XC3100A / L
Settings:	Command, Disable, Once
Default	Command

ResetTime

Removes INTERNAL RESET one clock cycle before or one clock cycle after the IOB becomes active.

When you specify the After setting, the outputs go active while all internal flip-flops are still being held in Reset. When you specify the Before setting, the internal logic becomes operational before the outputs go active.

Architectures:	XC3000A / L, XC3100A / L
Settings:	Before, After
Default	After

–g (Set Configuration—XC4000 and Spartan)

Note For Spartan-II -g options, see the "–g (Set Configuration—Virtex / -E / -II and Spartan-II Devices)" section.

This option specifies the startup timing and other bitstream options for the XC4000E / L, XC4000EX / XL / XV, and Spartan devices. Timing sequences are predefined startup defaults that use the following syntax.

```
bitgen -g timing_sequence
```

There are four valid startup sequences: Cclk_Nosync, Cclk_Sync, Uclk_Nosync, and Uclk_Sync. These startup sequences are described in the next section. For more information about startup timing, refer to *The Programmable Logic Data Book.*

The default startup sequence for the –g option is Cclk_Nosync. This startup sequence makes an XC4000 or Spartan device compatible with an XC3X00 device that is set for early Done and late Reset. Enter the following,

```
bitgen -g cclk_nosync
```

The –g option has sub-options that represent settings you use to set the configuration for an XC4000 or Spartan design. These options have the following syntax.

```
bitgen -g option:setting
```

For example, to enable Cyclic Redundancy Checking (CRC), use the following syntax.

```
bitgen -g crc:enable
```

The following sections describe the startup sequences for the –g option.

Startup Sequences and the –g Option

This section describes the four predefined startup sequences and their defaults; then it describes the options, their settings, and their defaults.

Note When mixing devices, the one with the latest "finished point" should be the master. The master stops clocking when it reaches the finished point. See *The Programmable Logic Data Book* for more information.

Cclk_Nosync

This is the default startup sequence for the –g option. Selecting this sequence causes the following defaults to take effect.

StartupClk:	Cclk
SyncToDone:	No
DoneActive:	C1
OutputsActive:	C2
GSRInactive:	C3

This startup sequence makes an XC4000, Spartan, or XC5200 device consistent with an XC3X00 device set for early Done and late Reset.

Cclk_Sync

Selecting this sequence causes the following defaults to take effect.

StartupClk:	Cclk
SyncToDone:	Yes
DoneActive:	C1
OutputsActive:	DI_PLUS_1
GSRInactive:	DI_PLUS_1

This startup sequence is the most consistent with the XC3X00 devices, since it synchronizes the release of GSR and I/Os to the external DoneIn signal. This startup sequence makes an XC4000 or Spartan device consistent with an XC3X00 device set for early Done and late Reset.

Uclk_Nosync

Selecting this sequence causes the following defaults to take effect

StartupClk:	Useclk
SyncToDone:	No
DoneActive:	U2
OutputsActive:	U3
GSRInactive:	U4

This startup sequence makes XC4000 or Spartan devices inconsistent with XC3X00 devices if they are in the same daisy chain, since the release of Done is synchronized to an external User Clock. There is no synchronization of I/Os or GSR to DoneIn.

Uclk_Sync

Selecting this sequence causes the following defaults to take effect.

StartupClk:	Userclk
SyncToDone:	Yes
DoneActive:	U2
OutputsActive:	D1_PLUS_1
GSRInactive:	D1_PLUS_2

This startup sequence makes XC4000 or Spartan devices inconsistent with XC3X00 devices if they are in the same daisy chain, since the release of Done is synchronized to an external User Clock. I/Os and GSR are synchronous to the clocks following DoneIn.

When using Uclk_Sync or Uclk_Nosync, you must provide a user clock to finish the configuration sequence. Without a user clock the FPGA will not configure.

Sub-Options for Startup Sequence (–g Option)

The sub-options available with the four startup sequences are described below. These sub-options use the **–g** *option*:*setting* syntax.

AddressLines

Determines the number of address lines (18 or 22) used for device configuration. The **22** setting activates four extra device pins as configuration address lines.

Architectures:	XC4000EX only (XC4000XL, XC4000XLA, and XC4000XV always have 22 active address lines)
Settings:	18, 22
Default	18

BSCAN_Config

When disabled, BSCAN_Config inhibits the BSCAN-based configuration after the device is successfully configured. This feature allows board testing without the risk of reconfiguring XLA devices by toggling the TCK/TMS/TDI/TDO lines.

Architectures:	XC4000XLA, XC4000XV, SpartanXL
Settings:	Disable, Enable
Default:	Enable

BSCAN_Status

When enabled, BSCAN_Status allows direct sensing of the DONE configuration state after performing a BSCAN-based configuration. Previously, there was no direct method for determining if a BSCAN-based configuration was successful.

Architectures:	XC4000XLA, XC4000XV, SpartanXL
Settings:	Disable, Enable
Default:	Disable

5V_Tolerant_IO

If set to On, this option allows a 3.3V device circuitry to tolerate 5V operation. For any device that operates on a mixed circuit environment with 3.3V and 5V, ensure that On is set. For any circuitry that operates exclusively on 3.3V, such as in a laptop computer, set the option to Off. The Off option reduces power consumption.

Architectures:	XC4000XLA, XC4000XV, SpartanXL / -II
Settings:	On, Off
Default:	On

ConfigRate

Selects the configuration clock rate. There are two choices: slow or fast. Slow is equivalent to 1 MHz, and fast is equivalent to 8 MHz (nominal).

Architectures:	XC4000E / L, XC4000EX / XL / XLA / XV, Spartan, and SpartanXL
Settings:	Slow, Fast
Default:	Slow

CRC

Enables or disables Cyclic Redundancy Checking (CRC) on a chip-by-chip basis during configuration.

Architectures:	XC4000E / L, XC4000EX / XL / XLA / XV, Spartan, and SpartanXL
Settings:	Enable, Disable
Default:	Enable

DoneActive

Selects the event that activates the FPGA Done signal. There are a maximum of four events that you can select from at one time. These events are Cclk edges or external (user) clock edges.

The actual options available at any time depend on the selections made for StartupClk and SyncToDone

Architectures:	XC4000E / L, XC4000EX / XL / XV / XLA, Spartan, and SpartanXL
Settings:	**C1** — first-Cclk rising edge after the length count is met. **C2** — second-Cclk rising edge after the length count is met. **C3** — third-Cclk rising edge after the length count is met. **C4** — fourth-Cclk rising edge after the length count is met. **U2** — second-valid-user-clock rising edge after C1. **U3** — third-valid-user-clock rising edge after C1. **U4** — fourth-valid-user-clock rising edge after C1.
Default:	C1

Valid settings for DoneActive are as follows.

StartupClk	SyncToDone	DoneActive
Cclk	Yes	C1, C2 or C3
Cclk	No	C1, C2, C3, or C4
UserClk	Yes	C1 or U2
UserClk	No	C1, U2, U3, or U4

DonePin

Enables or disables internal pull-up on the DONE pin. The Pullnone setting indicates there is no connection to the pull-up.

Architectures:	XC4000E / L, XC4000EX / XL / XLA / XV, Spartan, and SpartanXL
Settings:	Pullup, Pullnone
Default:	Pullup

ExpressMode

When enabled, ExpressMode creates a unique type of bitstream for configuration.

Architectures:	XC4000XLA, XC4000XV, SpartanXL
Settings:	Disable, Enable
Default:	Disable

GSRInactive

Selects the event that releases the internal set-reset to the latches and flip-flops. You can select one of nine events: a Cclk edge, an external (user) clock edge, or the external signal DoneIn. Only some of these events become options at one time depending on the combination of StartupClk and SyncToDone selected.

Architectures:	XC4000E / L, XC4000EX / XL / XLA / XV, Spartan, SpartanXL
Settings:	**C2** — second-Cclk rising edge after the length count is met. **C3** — third-Cclk rising edge after the length count is met. **C4** — fourth-Cclk rising edge after the length count is met. **U2** — second-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met). **U3** — third-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met). **U4** — fourth-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met). **DI** — when the DoneIn signal goes High. **DI_PLUS_1** — first Cclk or valid user clock rising edge (depending on selection of StartupClk) after DoneIn goes High. **DI_PLUS_2** — second Cclk or valid user clock rising edge (depending on selection of StartupClk) after DoneIn goes High.
Default:	C3

Valid settings for GSRInactive are as follows.

StartupClk	SyncToDone	GSRInactive
Cclk	Yes	C2, C3, DI, or DI_PLUS_1
Cclk	No	C2, C3, or C4
UserClk	Yes	U2, DI, DI_PLUS_1, or DI_PLUS_2
UserClk	No	U2, U3, or U4

Input

Sets the input threshold level for IOBs.

Architectures:	XC4000E / L, XC4000EX, Spartan
Settings:	TTL, CMOS
Default:	TTL

LC_Alignment

The LC_Alignment option determines how length count is calculated to control when the device changes from configuration to user operation. The two methods of calculating length count, DONE Alignment and Length Count Alignment, are discussed in the Configuration section of the The Programmable Logic Data Book. The FPGA Configuration Guidelines Application Note also contains length count information.

Architectures:	XC4000E / L, XC4000EX / XL / XLA / XV, Spartan, SpartanXL
Settings:	Length, DONE
Default	Length

M0Pin

Adds a pull-up or a pull-down to the M0 (Mode 0) pin. Selecting one option enables it and disables the others. The Pullnone setting indicates there is no connection to either the pull-up or the pull-down.

Architectures:	XC4000E / L, XC4000EX / XL / XLA / XV, SpartanXL
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullnone

M1 Pin

Adds a pull-up or a pull-down to the M1 (Mode 1) pin. Selecting one option enables it and disables the others. The Pullnone setting indicates there is no connection to either the pull-up or the pull-down

Architectures:	XC4000E / L, XC4000EX / XL / XLA / XV
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullnone

M2Pin

Adds a pull-up or a pull-down to the M2 (Mode 2) pin. Selecting one option enables it and disables the others. The Pullnone setting indicates there is no connection to either the pull-up or the pull-down.

Architectures:	XC4000E / L, XC4000EX / XL / XLA / XV
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullnone

Output

Sets the output level for IOBs,

Architectures:	XC4000E / L, XC4000EX, Spartan
Settings:	TTL, CMOS
Default:	TTL

OutputsActive

Selects the event that releases the I / O from 3-state condition and turns the configuration related pins operational. There are a maximum of four events that you can select from at one time. These events are selected from a group of Cclk edges, a group of external (user) clock edges, and the external signal DoneIn. The actual options available at any time depend on the selections made for StartupClk and SyncToDone.

Architectures: XC4000E / L, XC4000EX / XL / XLA / XV, Spartan, SpartanXL

Settings:
C2 — second-Cclk rising edge after the length count is met.
C3 — third-Cclk rising edge after the length count is met.
C4 — fourth-Cclk rising edge after the length count is met.
U2 — second-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met)
U3 — third-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met)
U4 — fourth-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met)
DI — when the DoneIn signal goes High
DI_PLUS_1 — first Cclk or valid user clock rising edge (depending on selection of StartupClk) after DoneIn goes High
DI_PLUS_2 — second Cclk or valid user clock rising edge (depending on selection of StartupClk) after DoneIn goes High

Default: C2

Valid settings for OutputsActive are as follows.

StartupClk	SyncToDone	OutputsActive
Cclk	Yes	C2, C3, DI, or DI_PLUS_1
Cclk	No	C2, C3, or C4
UserClk	Yes	U2, DI, DI_PLUS_1, or DI_PLUS_2
UserClk	No	U2, U3, or U4

PowerDown

Enables or disables internal pull-up on the PowerDown pin. The Pullnone setting indicates there is no connection to the pull-up.

Architectures:	SpartanXL
Settings:	Pullup, Pullnone
Default:	Pullup

ReadAbort

Enables or disables aborting the readback sequence during the read-back sequence.

Architectures:	XC4000E / L, XC4000EX / XL / XLA / XV, Spartan, SpartanXL
Settings:	Enable, Disable
Default:	Disable

ReadCapture

Enables or disables readback of configuration bitstream.

Architectures:	XC4000E / L, XC4000EX / XL / XLA / XV, Spartan, SpartanXL
Settings:	Enable, Disable
Default:	Disable

ReadClk

Sets the readback clock to be CClk or to a user-supplied clock (from a net inside the FPGA that is connected to the ‘i’ pin of the RDCLK schematic block).

Architectures:	XC4000E / L, XC4000EX / XL / XLA / XV, Spartan, SpartanXL
Settings:	Cclk (pin—see Note), Rdbk (user-supplied)
Default:	Cclk

Note In modes where CClk is an output, the pin is driven by the internal oscillator.

StartupClk

Selects a user-supplied clock or the internal Cclk for controlling the post-configuration startup phase of the FPGA initialization

Architectures:	XC4000E / L, XC4000EX / XL / XLA / XV, Spartan, SpartanXL
Settings:	Cclk (pin—see Note), UserClk (user-supplied)
Default:	Cclk

Note In modes where Cclk is an output, the pin is driven by the internal oscillator.

SyncToDone

Synchronizes the I / O startup sequence to the external DoneIn signal.

Architectures:	XC4000E / L, XC4000EX / XL / XLA / XV, Spartan, SpartanXL
Settings:	Yes, No
Default:	No

TdoPin

Adds a pull-up, a pull-down, or neither to the TDO pin (Test Data Out for Boundary Scan). Selecting one option enables it and disables the others. The Pullnone setting indicates there is no connection to either the pull-up or the pull-down.

Architectures:	XC4000E / L, XC4000EX / XL / XLA / XV, Spartan, SpartanXL
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullnone

–g (Set Configuration—XC5200 Devices)

The –g option has sub-options that represent settings you use to set the configuration for an XC5200 design. These options have the following syntax.

```
bitgen -g option:setting
```

For example, to enable Cyclic Redundancy Checking (CRC), use the following syntax.

```
bitgen –g crc:enable
```

The following sections describe the startup sequences for the –g option.

BSReconfig

Enable or disable reconfiguration via boundary scan.

Architectures:	XC5200
Settings:	Disable, Enable
Default:	Disable

BSReadback

Enable or disable reading back configuration data via boundary scan.

Architectures:	XC5200
Settings:	Disable, Enable
Default:	Disable

ConfigRate

Selects the configuration clock rate. There are three choices: slow, med, and fast. Slow is equivalent to .75 MHz, med is equivalent to 6 MHz, and fast is equivalent to 12 MHz (nominal).

Architectures:	XC5200
Settings:	Slow, Med, Fast
Default:	Slow

CRC

Enables or disables Cyclic Redundancy Checking (CRC) on a chip-by-chip basis during configuration.

Architectures:	XC5200
Settings:	Enable, Disable
Default:	Enable

Input

This option sets the FPGA design input-signal thresholds to TTL or CMOS level for interface capability. CMOS improves noise immunity and reduces static power consumption.

The special-purpose clock inputs, TCLKIN, BCLKIN, and $\overline{\text{PWRDN}}$ always require CMOS-level signals, even if the FPGA design input thresholds are specified as TTL compatible.

Architectures:	XC5200
Settings:	TTL, CMOS
Default:	TTL

DoneActive

Selects the event that activates the FPGA Done signal. There are a maximum of four events that you can select from at one time. These events are Cclk edges or external (user) clock edges.
The actual options available at any time depend on the selections made for StartupClk and SyncToDone.

Architectures:	XC5200
Settings:	**C1** — first-Cclk rising edge after the length count is met. **C2** — second-Cclk rising edge after the length count is met. **C3** — third-Cclk rising edge after the length count is met. **C4** — fourth-Cclk rising edge after the length count is met. **U2** — second-valid-user-clock rising edge after C1. **U3** — third-valid-user-clock rising edge after C1. **U4** — fourth-valid-user-clock rising edge after C1.
Default:	C1

Valid settings for DoneActive are as follows.

StartupClk	SyncToDone	DoneActive
Cclk	Yes	C1, C2 or C3
Cclk	No	C1, C2, C3, or C4
UserClk	Yes	C1 or U2
UserClk	No	C1, U2, U3, or U4

DonePin

Enables or disables internal pull-up on the DONE pin. The Pullnone setting indicates there is no connection to the pull-up.

Architectures:	XC5200
Settings:	Pullup, Pullnone
Default:	Pullup

GSRInactive

Selects the event that releases the internal set-reset to the latches and flip-flops. You can select one of nine events: a Cclk edge, an external (user) clock edge, or the external signal DoneIn.

Only some of these events become options at one time depending on the combination of StartupClk and SyncToDone selected.

Architectures:	XC5200
Settings:	**C2** — second-Cclk rising edge after the length count is met. **C3** — third-Cclk rising edge after the length count is met. **C4** — fourth-Cclk rising edge after the length count is met. **U2** — second-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met). **U3** — third-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met). **U4** — fourth-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met). **DI** — when the DoneIn signal goes High **DI_PLUS_1** — first Cclk or valid user clock rising edge (depending on selection of StartupClk) after DoneIn goes High **DI_PLUS_2** — second Cclk or valid user clock rising edge (depending on selection of StartupClk) after DoneIn goes High
Default:	C3

Valid settings for GSRInactive are as follows.

StartupClk	SyncToDone	GSRInactive
Cclk	Yes	C2, C3, DI, or DI_PLUS_1
Cclk	No	C2, C3, or C4
UserClk	Yes	U2, DI, DI_PLUS_1, or DI_PLUS_2
UserClk	No	U2, U3, or U4

Input

Sets the input threshold level for IOBs.

Architectures:	XC5200
Settings:	TTL, CMOS
Default:	TTL

LC_Alignment

The LC_Alignment option determines how length count is calculated to control when the device changes from configuration to user operation. The two methods of calculating length count, DONE Alignment and Length Count Alignment, are discussed in the Configuration section of *The Programmable Logic Data Book*. The *FPGA Configuration Guidelines* Application Note also contains length count information.

Architectures:	XC5200
Settings:	Length, DONE
Default:	Length

OscClk

Determines whether the XC5200 oscillator is driven by the internal 16-MHz clock (CClk setting) or by a user clock (UserClk setting). If you specify UserClk, the clock must be connected to the OSC.CK pin of the device's OSC component.

Architectures:	XC5200
Settings:	UserClk, CClk
Default:	Cclk

OutputsActive

Selects the event that releases the I/O from 3-state condition and turns the configuration related pins operational. There are a maximum of four events that you can select from at one time. These events are selected from a group of Cclk edges, a group of external (user) clock edges, and the external signal DoneIn.

The actual options available at any time depend on the selections made for StartupClk and SyncToDone.

Architectures:	XC5200
Settings:	**C2** — second-Cclk rising edge after the length count is met. **C3** — third-Cclk rising edge after the length count is met. **C4** — fourth-Cclk rising edge after the length count is met. **U2** — second-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met). **U3** — third-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met). **U4** — fourth-valid-user-clock rising edge after C1 (first-Cclk rising edge after length count is met). **DI** — when the DoneIn signal goes High **DI_PLUS_1** — first Cclk or valid user clock rising edge (depending on selection of StartupClk) after DoneIn goes High **DI_PLUS_2** — second Cclk or valid user clock rising edge (depending on selection of StartupClk) after DoneIn goes High
Default:	C2

Valid settings for OutputsActive are as follows.

StartupClk	SyncToDone	OutputsActive
Cclk	Yes	C2, C3, DI, or DI_PLUS_1
Cclk	No	C2, C3, or C4
UserClk	Yes	U2, DI, DI_PLUS_1, or DI_PLUS_2
UserClk	No	U2, U3, or U4

ProgPin

Enables or disables internal pull-up on the $\overline{\text{PROGRAM}}$ pin. The pull-up affects the pin after configuration. The Pullnone setting indicates there is no connection to the pull-up.

Architectures:	XC5200
Settings:	Pullup, Pullnone
Default:	Pullup

ReadAbort

Enables or disables aborting the readback sequence during the readback sequence.

Architectures:	XC5200
Settings:	Enable, Disable
Default:	Disable

ReadCapture

Enables or disables readback of configuration bitstream.

Architectures:	XC5200
Settings:	Enable, Disable
Default:	Disable

ReadClk

Sets the readback clock to be CClk or to a user-supplied clock (from a net inside the FPGA that is connected to the ‘i’ pin of the RDCLK schematic block).

Architectures:	XC5200
Settings:	Cclk (pin—see Note), Rdbk (user-supplied)
Default:	Cclk

Note In modes where CClk is an output, the pin is driven by the internal oscillator.

StartupClk

Selects a user-supplied clock or the internal Cclk for controlling the post-configuration startup phase of the FPGA initialization.

Architectures:	XC5200
Settings:	Cclk (pin—see Note), UserClk (user-supplied)
Default:	Cclk

Note In modes where CClk is an output, the pin is driven by the internal oscillator.

SyncToDone

Synchronizes the I/O startup sequence to the external DoneIn signal.

Architectures:	XC5200
Settings:	Yes, No
Default:	No

–g (Set Configuration—Virtex/-E/-II and Spartan-II Devices)

The –g option has sub-options that represent settings you use to set the configuration for a Virtex/-E/-II or Spartan-II design. These options have the following syntax.

```
bitgen -g option:setting
```

For example, to enable Readback, use the following syntax.

```
bitgen -g Readback
```

The following sections describe the startup sequences for the –g option.

ReadBack

This option allows you to perform Readback by the creating the necessary bitstream.

Compress

This option uses the multiple frame write feature in the bitstream to reduce the size of the bitstream, not just the .bit file. Using the Compress option does not guarantee that the size of the bitstream will shrink.

ConfigRate

Virtex/-E/-II and Spartan-II use an internal oscillator to generate the configuration clock, CCLK, when configuring in a master mode. Use the configuration rate option to select the rate for this clock.

Architectures:	Virtex/-E/-II, Spartan-II
Settings:	To find out settings, enter **bitgen -h virtex**. Values are in MHz. The default is 4.
Default:	The default is the first item listed with **bitgen -h virtex** command.

Gclkdel0, Gclkdel1, Gclkdel2, Gclkdel3

Use these options to add delays to the global clocks. *You should not use this option unless instructed by Xilinx.*

Architectures:	Virtex/-E/-II, Spartan-II
Settings:	11111, *binary string*
Default:	11111

StartupClk

The startup sequence following the configuration of a device can be synchronized to either Cclk, a User Clock, or the JTAG Clock. The default is Cclk.

- Cclk

  Enter Cclk to synchronize to an internal clock provided in the FPGA device.

- UserClk

  Enter UserClk to synchronize to a user-defined signal connected to the CLK pin of the STARTUP symbol.

- Jtag Clock

  Enter JtagClk to synchronize to the clock provided by JTAG. This clock sequences the TAP controller which provides the control logic for JTAG.

Architectures:	Virtex/-E/-II, Spartan-II
Settings:	Cclk (pin—see Note), UserClk (user-supplied), JtagCLK
Default:	Cclk

Note In modes where Cclk is an output, the pin is driven by an internal oscillator.

PowerupClk

Selects which clock to synchronize to at the end of power up.

Architectures:	Virtex/-E/-II, Spartan-II
Settings:	IntOsc, UserClk, CClk
Default:	IntOsc

CclkPin

Adds an internal pull-up to the Cclk pin. The Pullnone setting disables the pullup.

Architectures:	Virtex/-E/-II, Spartan-II
Settings:	Pullnone, Pullup
Default:	Pullup

DonePin

Adds an internal pull-up to the DonePin pin. The Pullnone setting disables the pullup.

Use this option only if you are planning to connect an external pull-up resistor to this pin. The internal pull-up resistor is automatically connected if you do not use this option.

Architectures:	Virtex/-E/-II, Spartan-II
Settings:	Pullup, Pullnone
Default:	Pullup

DrivePDStatusPin

Enables this output power-down status pin.

Architectures:	Spartan-II only
Settings:	Yes, No
Default:	Yes

M0Pin

The M0 pin is used to determine the configuration mode. Adds an internal pull-up, pull-down or neither to the M0 pin. The following settings are available. The default is PullUp. Select Pullnone to disable both the pull-up resistor and pull-down resistor on the M0 pin.

Architectures:	Virtex/-E/-II, Spartan-II
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullup

M1Pin

The M1 pin is used to determine the configuration mode. Adds an internal pull-up, pull-down or neither to the M1 pin. The following settings are available. The default is PullUp.

Select **Pullnone** to disable both the pull-up resistor and pull-down resistor on the M1 pin.

Architectures:	Virtex/-E/-II, Spartan-II
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullup

M2Pin

The M2 pin is used to determine the configuration mode. Adds an internal pull-up, pull-down or neither to the M2 pin. The default is PullUp. Select Pullnone to disable both the pull-up resistor and pull-down resistor on the M2 pin.

Architectures:	Virtex/-E/-II, Spartan-II
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullup

PDStatusPin

Adds an internal pull-up to the PDStatusPin pin. The Pullnone setting disables the pullup.

Use this option only if you are planning to connect an external pull-up resistor to this pin.

Architectures:	Spartan-II only
Settings:	Pullup, Pullnone
Default:	Pullnone

PowerdownPin

Adds an internal pull-up to the input PowerdownPin pin. The Pullnone setting disables the pullup.

Use this option only if you are planning to connect an external pull-up resistor to this pin.

Architectures:	Spartan-II only
Settings:	Pullup, Pullnone
Default:	Pullnone

ProgPin

Adds an internal pull-up to the ProgPin pin. The Pullnone setting - disables the pullup. The pull-up affects the pin after configuration.

Architectures:	Virtex/-E/-II, Spartan-II
Settings:	Pullup, Pullnone
Default:	Pullnone

TckPin

Adds a pull-up, a pull-down or neither to the TCK pin, the JTAG test clock. Selecting one setting enables it and disables the others. The Pullnone setting indicates there is no connection to either the pull-up or the pull-down.

Architectures:	Virtex/-E/-II, Spartan-II
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullup

TdiPin

Adds a pull-up, a pull-down, or neither to the TDI pin, the serial data input to all JTAG instructions and JTAG registers. Selecting one setting enables it and disables the others. The Pullnone setting indicates there is no connection to either the pull-up or the pull-down.

Architectures:	Virtex/-E/-II, Spartan-II
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullup

TdoPin

Adds a pull-up, a pull-down, or neither to the TdoPin pin, the serial data output for all JTAG instruction and data registers. Selecting one setting enables it and disables the others. The Pullnone setting indicates there is no connection to either the pull-up or the pull-down.

Architectures:	Virtex/-E/-II, Spartan-II
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullup

TmsPin

Adds a pull-up, pull-down, or neither to the TMS pin, the mode input signal to the TAP controller. The TAP controller provides the control logic for JTAG. Selecting one setting enables it and disables the others. The Pullnone setting indicates there is no connection to either the pull-up or the pull-down

Architectures:	Virtex/-E/-II, Spartan-II
Settings:	Pullup, Pulldown, Pullnone
Default:	Pullup

UnusedPin

Adds a pull-up, a pull-down, or neither to the UnusedPin, the serial data output for all JTAG instruction and data registers. Selecting one setting enables it and disables the others. The Pullnone setting indicates there is no connection to either the pull-up or the pull-down.

The following settings are available. The default is PullDown.

Architectures:	Virtex/-E/-II, Spartan-II
Settings:	Pullup, Pulldown, Pullnone
Default:	Pulldown

GSR_cycle

Selects the Startup phase that releases the internal set-reset to the latches, flip-flops, and BRAM output latches. The Done setting releases GSR when the DoneIn signal is High. DoneIn is either the value of the Done pin or a delayed version if DonePipe=Yes

Architectures:	Virtex/-E/-II, Spartan-II
Settings:	Done, 1, 2, 3, 4, 5, 6, Keep
Default:	6

Keep should only be used when partial reconfiguration is going to be implemented. Keep prevents the configuration state machine from asserting control signals that could cause the loss of data.

GWE_cycle

Selects the Startup phase that asserts the internal write enable to flip-flops, LUT RAMs, and shift registers. It also enables the BRAMs. Before the Startup phase both BRAM writing and reading are disabled.The Done setting asserts GWE when the DoneIn signal is High. DoneIn is either the value of the Done pin or a delayed version if DonePipe=Yes. The Keep setting is used to keep the current value of the GWE signal

Architectures:	Virtex/-E/-II, Spartan-II
Settings:	Done, 1, 2, 3, 4, 5, 6, Keep
Default:	6

GTS_cycle

Selects the Startup phase that releases the internal tristate control to the IO buffers. The Done setting releases GTS when the DoneIn signal is High. DoneIn is either the value of the Done pin or a delayed version if DonePipe=Yes

Architectures:	Virtex/-E/-II, Spartan-II
Settings:	Done, 1, 2, 3, 4, 5, 6, Keep
Default:	5

LCK_cycle

Selects the Startup phase to wait until DLLs lock. If NoWait is selected, the Startup sequence does not wait for DLLs.

Architectures:	Virtex/-E/-II, Spartan-II
Settings:	0,1, 2, 3, 4, 5, 6, NoWait
Default:	NoWait

DONE_cycle

Selects the Startup phase that activates the FPGA Done signal. Done is delayed when DonePipe=Yes.

Architectures:	Virtex/-E/-II, Spartan-II
Settings:	1, 2, 3, 4, 5, 6
Default:	4

Persist

This option is needed for Readback and Partial Reconfiguration using the SelectMAP configuration pins. If Persist is set to Yes, the pins used for SelectMAP mode are prohibited for use as user IO. Refer to the data sheet for a description of SelectMAP mode and the associated pins.

Architectures:	Virtex/-E/-II, Spartan-II
Settings:	No, Yes
Default:	No

DriveDone

This option actively drives CFG_DONE (Done) high as opposed to using pullup.

Architectures:	Virtex/-E/-II, Spartan-II
Settings:	No, Yes
Default:	No

DonePipe

This option is intended for use with FPGAs being set up in a high-speed daisy chain configuration.When set to Yes, the FPGA waits on the CFG_DONE (DONE) pin to go High and then waits for the first clock edge before moving to the Done state.

Architectures:	Virtex/-E/-II, Spartan-II
Settings:	No, Yes
Default:	No

Security

Selecting Level1 disables Readback. Selecting Level2 disables Readback and Partial Reconfiguration.

Architectures:	Virtex/-E/-II, Spartan-II
Settings:	None, level1, Level2
Default:	None

UserID

You can enter up to an 8-digit hexadecimal code in the User ID register. You can use the register to identify implementation revisions.

DebugBitstream

If the device does not configure correctly, you can debug the bitstream using this option. The values allowed for the DebugBitstream option are No and Yes.

Architectures:	Virtex/-E/-II, Spartan-II
Values:	No, Yes

In addition to a standard bitstream, a debug bitstream offers the following features.

- Writes 32 0s to the LOUT register after the synchronization word
- Loads each frame individually
- Performs a cyclical redundancy check (CRC) after each frame
- Writes a column number to the LOUT register after each frame

–h or –help (Command Usage)

–h *architecture*

Displays a usage message for BitGen. The usage message displays all of the available options for BitGen operating on the specified *architecture*.

–j (No BIT File)

Do not create a bitstream file (.bit file). This option is generally used when you want to generate a report without producing a bitstream. For example, if you wanted to run DRC without producing a bitstream file, you would use the -j option.

Note The .msk or .rbt files may still be created.

–l (Create a Logic Allocation File)

This option creates an ASCII logic allocation file (*design*.**ll**) for the selected design. The logic allocation file indicates the bitstream position of latches, flip-flops, and IOB inputs and outputs.

In some applications, you may want to observe the contents of the FPGA internal registers at different times. The file created by the –l option helps you identify which bits in the current bitstream represent outputs of flip-flops and latches. Bits are referenced by frame and bit number within the frame.

The Hardware Debugger uses the **design.ll** file to locate signal values inside a readback bitstream.

–m (Generate a Mask File)

Creates a mask file. This file is used to compare relevant bit locations for executing a readback of configuration data contained in an operating FPGA.

–n (Save a Tied design)

This command is used with the –t option (described below) to save the tied NCD file as *_file_name*.ncd (note the underscore in front of the file name). The tied design file is placed in the same directory as the output file. It has the same root name as the output file with an .ncd extension. If you do not specify an output file, the tied design file is placed in the input file's directory and is named *_file_name*.ncd, where *_file_name* is the root name of the input file. Use TRACE to run timing analysis on the tied design. You can also use the FPGA Editor to check the effects of the tiedown. This option is not supported for Virtex/-E/-II or Spartan-II.

–t (Tie Unused Interconnect)

This option causes all unused interconnect to be tied to a logic low or to a known level, keeping internal noise and power consumption to a minimum. When you use the –t option, DRC runs first (before tiedown). BitGen terminates if any DRC error occurs. A DRC warning does not cause the bitstream generation program to abort, but it may cause tiedown to fail.

After DRC, the –t option does the following.

- Ties all possible unused interconnect to tie sites or unused CLB outputs and configures those outputs with a logic low (F=0 or G=0)
- Attempts to tie any remaining interconnect to CLB outputs which have not been designated as critical
- Attempts to tie remaining interconnect to the global or to the auxiliary clock buffer outputs if unused (only in conjunction with the -a option)

The only condition under which tie will add interconnect to a "critical" net is if you use the –u option (allowing interconnect to be added to critical nets as a "last resort"). A "critical" net is one with a priority greater than 3.

The –t option does not add an XC4000 or XC5200 tristate buffer input (I) pin or tristate (T) pin to a net.

When you add interconnect to used CLB or buffer outputs, delays may be added on any net to which the outputs are connected. To prevent the added delay, assign the net a priority greater than 3. You can do this through the physical constraints file or through the FPGA Editor. See the PRIORITIZE physical constraint in the "Attributes, Constraints, and Carry Logic" chapter of the *Libraries Guide*. Note that flagging too many nets as critical could cause the tiedown to fail. When an interconnect is tied to a user-defined net, you get a message giving the number of nodes added to the net. Delay characteristics for the net associated with that source may change. (Only in conjunction with the -a option)

When certain pins cannot be tied, you receive a warning message supplying information about the design's untied interconnect.

To remove the obstacles that have caused tiedown to fail, look carefully at nets close to an untied PIP. An input pin could have multiple input PIPs, and all of them could source the pin. If each of these PIPs is associated with a critical net, they are not used, and the input pin is left untied. To correct the problem, make one of the nets "non-critical." Do this by removing the PRIORITIZE constraint from the net in the PCF file or in the FPGA Editor. Then run TRACE (the timing analysis program) and evaluate any delay that might have been added to the net. (Only in conjunction with the -a option)

If you use the –n option, the tied design is saved in a file *_file_name*.ncd (note the underscore before the file name). You can load the file into the FPGA Editor and examine the results of tiedown. You can look at all of the original nets that have been affected by tiedown and the net delays before and after tiedown.

Like unused internal interconnect, unused external I/O pins on the chip must also have defined signal levels, that is, they must not be in a floating condition. In XC4000E/EX FPGAs, unused IOBs are automatically pulled HIGH with pull-up resistors.

Partial tiedown is the new default. Tiedown will print the number of untied nodes and then continue. See the -a option also. Partial tiedown *never* ties to user signals.

This option is not supported for Virtex/-E/-II or Spartan-II.

–u (Use Critical Nets Last)

Because of possible added delay, tiedown does not add interconnect to any net that has been assigned a priority greater than 3. This option allows interconnect to be added to critical nets as a “last resort.”

This option is not supported for Virtex/-E/-II or Spartan-II.

–w (Overwrite Existing Output File)

Enables you to overwrite an existing BIT, LL, MSK, or RBT output file.

Chapter 17

PROMGen

This program is compatible with the following Xilinx devices.

- Spartan / XL / -II
- Virtex / E / -II
- XC4000E / L / EX / XL / XV / XLA
- XC3000A / L
- XC3100A / L
- XC5200

This chapter describes PROMGen. The chapter contains the following.

- “PROMGen”
- “PROMGen Syntax”
- “PROMGen Files”
- “PROMGen Options”
- “Examples” section

PROMGen

PROMGen formats a BitGen-generated configuration bitstream (BIT) file into a PROM format file. The PROM file contains configuration data for the FPGA device. PROMGen converts a BIT file into one of three PROM formats: MCS-86 (Intel), EXORMAX (Motorola), or TEKHEX (Tektronix). It can also generate a Hex file format.

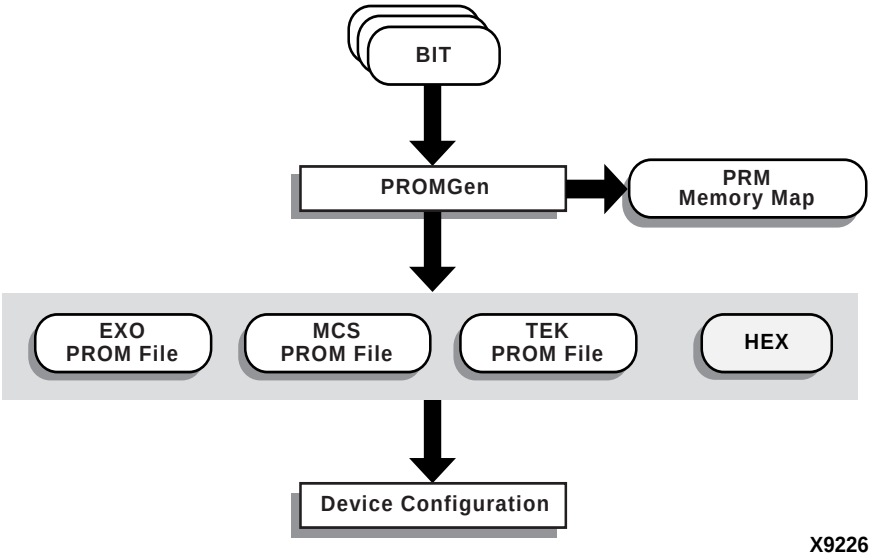


Figure 17-1 PROMGen

There are two functionally equivalent versions of PROMGen. There is a stand-alone version you can access from an operating system prompt. There is also an interactive version, called the PROM File Formatter, that you can access from inside the Design Manager for Alliance or the Project Manager in Foundation. This chapter first describes the stand-alone version; the interactive version is described in the *PROM File Formatter Guide.*

You can also use PROMGen to concatenate bitstream files to daisy-chain FPGAs.

Note If the destination PROM is one of the Xilinx Serial PROMs, you are using a Xilinx PROM Programmer, and the FPGAs are not being daisy-chained, it is not necessary to make a PROM file. See the *Hardware User Guide* for more information about daisy-chained designs

PROMGen Syntax

To start PROMGen from the operating system prompt, use the following syntax.

promgen [*options*]

Options can be any number of the options listed in the "PROMGen Options" section. Separate multiple options with spaces.

PROMGen Files

This section describes the PROMGen input and output files.

Input Files

The input to PROMGEN consists of BIT files— one or more bitstream files. BIT files contain configuration data for an FPGA design.

Output Files

Output from PROMGEN consists of the following files.

- PROM files—The file or files containing the PROM configuration information. Depending on the PROM file format your PROM programmer uses, you can output a TEK, MCS, or EXO file. If you are using a microprocessor to configure your devices, you can output a HEX file, which contains a hexadecimal representation of the bitstream.
- PRM file—The PRM file is a PROM image file. It contains a memory map of the output PROM file. The file has a .prm extension.

Bit Swapping in PROM Files

PROMGen produces a PROM file in which the bits within a byte are swapped compared to the bits in the input BIT file. Bit swapping (also called "bit mirroring") reverses the bits within each byte, as shown in the following figure.

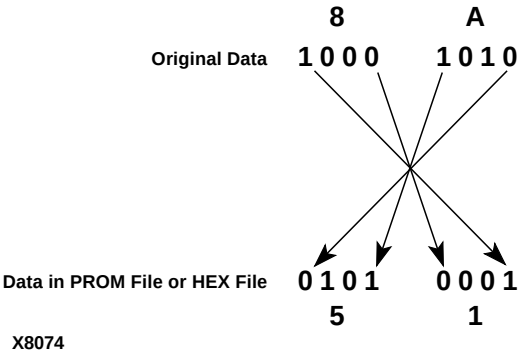


Figure 17-2 Bit Swapping

In a bitstream contained in a BIT file, the Least Significant Bit (LSB) is always on the left side of a byte. But when a PROM programmer or a microprocessor reads a data byte, it identifies the LSB on the right side of the byte. In order for the PROM programmer or microprocessor to read the bitstream correctly, the bits in each byte must first be swapped so they are read in the correct order.

In this release of the Xilinx Development System, the bits are swapped for all of the PROM formats: MCS, EXO, and TEK. For a HEX file output, bit swapping is on by default, but it can be turned off by entering a –b PROMGen option that is available only for HEX file format.

PROMGen Options

This section describes the options that are available for the PROMGen command.

–b (Disable Bit Swapping—HEX Format Only)

This option only applies if the –p option specifies a HEX file for the output of PROMGen. By default (no –b option), bits in the HEX file are swapped compared to bits in the input BIT files. If you enter a –b option, the bits are not swapped. Bit swapping is described in the “Bit Swapping in PROM Files” section.

–c (Checksum)

promgen –c

The –c option generates a checksum value appearing in the .prm file. This value should match the checksum in the prom programmer. Use this option to verify that correct data was programmed into the prom.

–d (Load Downward)

promgen –d *hexaddress0 filename filename...*

This option loads one or more BIT files from the starting address in a downward direction. Specifying several files after this option causes the files to be concatenated in a daisy chain. You can specify multiple –d options to load files at different addresses. You must specify this option immediately before the input bitstream file.

Here is the multiple file syntax.

promgen –d *hexaddress0 filename filename...*

Here is the multiple –d options syntax.

promgen –d *hexaddress1 filename* **-d** *hexaddress2 filename...*

–f (Execute Commands File)

–f *command_file*

The –f option executes the command line arguments in the specified *command_file*. For more information on the –f option, see the "–f Option" section of the "Introduction" chapter.

–help (Command Help)

This option displays help that describes the PROMGen options.

–l option (Disable Length Count)

promgen –l

The –l option disables the length counter in the FPGA bitstream. It is valid only for SpartanXL, 4000EX, 4000XL, 4000XV, and 4000XLA devices. Use this option when chaining together bitstreams exceeding the 24 bit limit imposed by the length counter.

–n (Add BIT FIles)

-n *file1*[**.bit**] *file2*[**.bit**]...

This option loads one or more BIT files up or down from the next available address following the previous load. The first –n option *must* follow a –u or –d option because -n does not establish a direction. Files specified with this option are not daisy-chained to previous files. Files are loaded in the direction established by the nearest prior –u, –d, or –n option.

The following syntax shows how to specify multiple files. When you specify multiple files, PROMGen daisy-chains the files.

promgen –d *hexaddress file0* **–n** *file1 file2...*

The syntax for using multiple –n options follows. Using this method prevents the files from being daisy-chained.

promgen –d *hexaddress file0* **–n** *file1* **-n** *file2...*

–o (Output File Name)

-o *file1*[*.ext*] *file2*[*.ext*]...

This option specifies the output file name of a PROM if it is different from the default. If you do not specify an output file name, the PROM file has the same name as the first BIT file loaded.

ext is the extension for the applicable PROM format.

Multiple file names may be specified to split the information into multiple files. If only one name is supplied for split PROM files (by you or by default), the output PROM files are named *file_#*.ext, where *file* is the base name, # is 0, 1, etc., and *ext* is the extension for the applicable PROM format.

promgen –d *hexaddress file0* **–o** *filename*

–p (PROM Format)

-p {**mcs** | **exo** | **tek** | hex}

This option sets the PROM format to one of the following: MCS (Intel MCS86), EXO (Motorola EXORMAX), TEK (Tektronix TEKHEX). The option may also produce a HEX file, which is a hexadecimal representation of the configuration bitstream used for microprocessor

downloads. If specified, the –p option must precede any –u, –d, or –n options. The default format is MCS.

–r (Load PROM File)

-r *promfile*

This option reads an existing PROM file as input instead of a BIT file. All of the PROMGen output options may be used, so the –r option can be used for splitting an existing PROM file into multiple PROM files or for converting an existing PROM file to another format.

–s (PROM Size)

-s *promsize1 promsize2...*

This option sets the PROM size in kilobytes. The PROM size must be a power of 2. The default value is 64 kilobytes. The –s option must precede any –u, –d, or –n options.

Multiple *promsize* entries for the –s option indicates the PROM will be split into multiple PROM files.

Note PROMGen PROM sizes are specified in bytes. *The Programmable Logic Data Book* specifies PROM sizes in bits for Xilinx serial PROMs (see –x option).

–u (Load Upward)

-u *hexaddress0 filename1 filename2...*

This option loads one or more BIT files from the starting address in an upward direction. When you specify several files after this option, PROMGen concatenates the files in a daisy chain. You can load files at different addresses by specifying multiple –u options.

This option must be specified immediately before the input bitstream file.

–x (Specify Xilinx PROM)

-x *xilinx_prom1 xilinx_prom2...*

The –x option specifies one or more Xilinx serial PROMs for which the PROM files are targeted. Use this option instead of the –s option if you know the Xilinx PROMs to use.

Multiple *xilinx_prom* entries for the –x option indicates the PROM will be split into multiple PROM files.

Examples

To load the file test.bit up from address 0x0000 in MCS format, enter the following information at the command line.

```
promgen –u 0 test
```

To daisy-chain the files test1.bit and test2.bit up from address 0x0000 and the files test3.bit and test4.bit from address 0x4000 while using a 32K PROM and the Motorola EXORmax format, enter the following information at the command line.

```
promgen –s 32 –p exo –u 00 test1 test2 –u 4000 test3
test4
```

To load the file test.bit into the PROM programmer in a downward direction starting at address 0x400, using a Xilinx XC1718D PROM, enter the following information at the command line.

```
promgen –x xc1718d –d 0x400 test
```

To specify a PROM file name that is different from the default file name enter the following information at the command line.

```
promgen options filename –o newfilename
```

Chapter 18

NGDAnno

This program is compatible with the following families.

- Spartan / XL / -II
- Virtex / -E / -II
- XC4000E / L / EX / XL / XV / XLA
- XC3000A / L
- XC3100A / L
- XC5200

This chapter describes the NGDAnno program. The chapter contains the following sections.

- "NGDAnno"
- "NGDAnno Syntax"
- "NGDAnno Files"
- "NGDAnno Options"
- "Dedicated Global Signals in Back-Annotation Simulation"
- "Hierarchy Changes in Annotated Designs"
- "Guaranteed Setup and Hold Check"

NGDAnno

The back-annotation process generates a generic timing simulation model. In the Xilinx Development System, NGDAnno back-annotates timing information using an NCD file and optionally, an NGM file. The NCD file, the output of MAP or PAR, represents the physical design. The NGM file, the output of MAP, represents the logical

design. NGDAnno performs either physical back-annotation or logical back-annotation depending on whether an NGM file is supplied as input.

If you do *not* supply an NGM file, NGDAnno performs physical back-annotation. It produces a simulation model based on the physical implementation only.

If you supply an NGM file, NGDAnno performs logical back-annotation. It distributes timing information associated with placement, routing, and block configuration from the physical NCD design file onto the logical design represented in the NGM file. If timing delays *cannot* be back-annotated to the logical model, NGDAnno inserts a physical model in place of the logical model. For example, NGDAnno might insert a LUT in place of a group of AND gates. The ARF file shows which instance or net names were replaced. For more information, see the "Hierarchy Changes in Annotated Designs" section.

Note If you make logical changes to an NCD design in the FPGA Editor and change the functional behavior of your design, NGDAnno cannot correlate the changed objects in the physical design with the objects in the logical design. NGDAnno recreates the entire NGA design from the NCD file. A warning indicates that the NCD file is no longer synchronized with the NGM file and that a new NGA file has been created from the NCD file.

For both logical and physical back-annotation, NGDAnno outputs an annotated logical design that has a .nga (Native Generic Annotated) extension. The NGA file is input to the appropriate netlist writer (NGD2EDIF, NGD2VHDL, or NGDVER). The netlist writer converts the back-annotated file in Xilinx format into netlist format for simulation.

In addition to back-annotating a fully routed design, you can perform back-annotation at other stages in the design flow. If you want to verify that the logic is correct *before* you map your design, you can use the data in an unmapped NGD file as input to the NGD2EDIF, NGD2VER, or NGD2VHDL program and run a simulation program on the resulting netlist. If you want to simulate with components, and not route delays, you can run back-annotation on the unrouted NCD file generated by the MAP program.

The back-annotation flow is shown in the following figure.

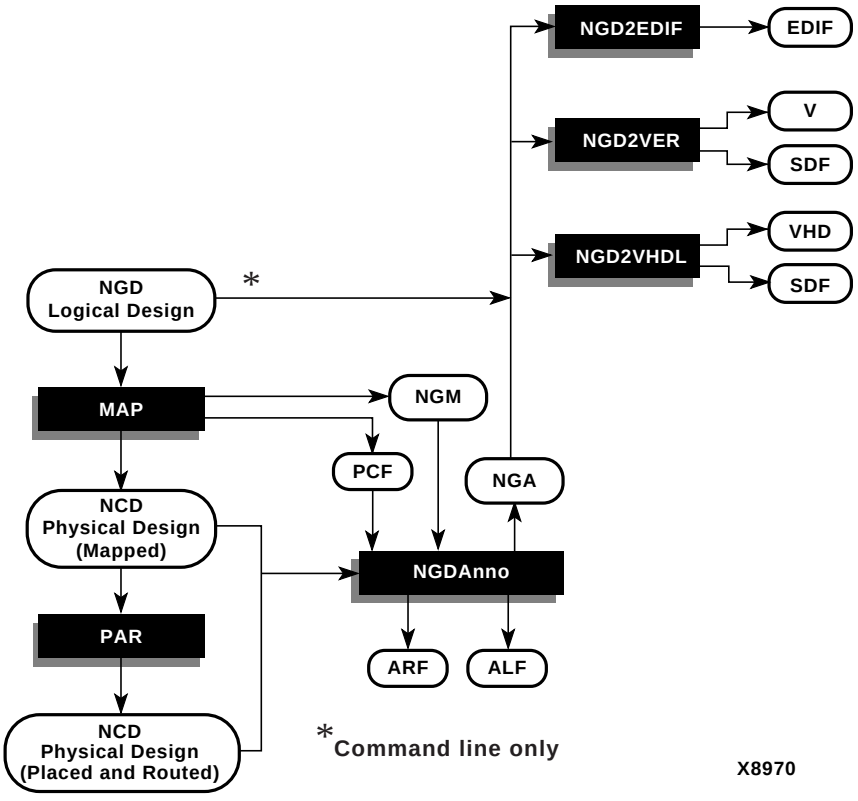


Figure 18-1 Back-Annotation Flow

You can run back-annotation by invoking NGDAnno and netlist writer programs from the UNIX or DOS command line or from the Design Manager / Flow Engine. Command line usage is explained in this chapter and in the netlist writer chapters. To use the Design Manager / Flow Engine for any of the programs, see the *Design Manager/Flow Engine Guide*.

NGDAnno Syntax

To perform back-annotation from the UNIX or DOS command line, enter the following.

ngdanno [*options*] *ncd_file*[**.ncd**] [*ngm_file*[**.ngm**]]

ncd_file is the input NCD (physical design file). If you specify an NCD file on the command line without specifying an NGM file, NGDAnno performs physical back-annotation and the NGA file is generated from the NCD file. The NGA file contains annotated information about the physical implementation only.

ngm_file is an optional NGM file—a design file produced by MAP that contains information about the logical design and information about how the logical design corresponds to the physical design. If you specify an NGM file, NGDAnno attempts to annotate physical information onto the logical netlist. The NGM file is useful in the following ways.

- For schematic-based designs, the NGM file can help regain net names lost in mapping.
- For HDL synthesis-based designs, the NGM file can help recover the original design hierarchy.

If you do not specify an NGA file with the –o option (described in the "NGDAnno Options" section), an NGA file is generated in the same directory as the NCD. The NGA file has the same root name as the NCD file. For example, the following command generates an NGA file named mydesign.nga.

```
ngdanno mydesign.ncd
```

NGDAnno Files

This section describes the NGDAnno input and output files.

Input Files

Input to the NGDAnno program is the following.

- NCD file—This physical design file may be mapped only, partially or fully placed, and partially or fully routed.
- NGM file (optional but recommended)—This mapped NGD file is created by the MAP program. This file contains a logical and

physical netlist and the correlation of blocks and pins between the two.

- PCF file (optional)—This is a physical constraints file.

Output Files

Output from the NGDAnno program is the following.

- NGA file—This is a back-annotated NGD file.

  Note NGA files generated in previous releases cannot be used with the netlist writers (NGD2EDIF, NGD2VHDL, or NGDVER) in this release. You must rerun NGDAnno to generate an NGA file for use with a netlist writer.

- ALF file—This annotation log file contains information about the NGDAnno run, including information on logical annotation failures. The ALF file has the same root name as the output NGA file and an .alf extension. The file is written into the same directory as the output NGA file.

- ARF file—This annotation report file contains information about lost instance or net names. The ARF file has the same root name as the output NGA file and an .arf extension. The file is written into the same directory as the output NGA file.

  This report file is only generated when you use the –report option. See the "–report (Generate Hierarchy Loss Report)" section for more information.

NGDAnno Options

This section contains descriptions of NGDAnno command line options.

–f (Execute Commands File)

-f *command_file*

The –f option executes the command line arguments in the specified *command_file*. For more information on the –f option, see the "–f Option" section of the "Introduction" chapter.

–module (Physical Simulation of Active Module)

The –module option locates the active module in your NCD file and creates an NGA file based on this module. NGDAnno constructs the top-level interface in the NGA file based only on the active module's interface signals.

To use this option you must specify an NCD file that contains an active module. To specify an active module, use the –modular module option with NGDBuild as described in the "–modular module (Active Module Implementation)" section of the "NGDBuild" chapter.

For more information on modular design, see `http://support.xilinx.com/xapp/xapp404.pdf`.

Note You cannot use the –module option with an NGM file or with an NCD file that does not contain an active module. In these cases, NGDAnno issues an error.

–o (Output File Name)

–o *out_file*[**.nga**]

The –o option specifies the output design file in NGA format. The .nga extension is optional. The output file name and its location are determined in the following ways.

- If you do not specify an output file name with the –o option, the output file has the same name as the input NCD file, with an .nga extension. The file is placed in the input NCD file's directory.
- If you specify an output file name with no path specifier (for example, **cpu_dec.nga** instead of **/home/designs/cpu_dec.nga**), the NGA file is placed in the current working directory.
- If you specify an output file name with a full path specifier (for example, **/home/designs/cpu_dec.nga**), the output file is placed in the specified directory.

If the output file already exists, it is overwritten with the new NGA file.

–p (PCF File)

-p *pcf_file*.**pcf**

The –p option allows you to specify a PCF (Physical Constraints) file as input to NGDAnno. You only need to specify a constraints file if it contains the following.

- Level information (CMOS or TTL) for IOBs in a 4000E or 4000EX design
- Prorating constraints

Prorating constraints and prorated delays are described in the "Prorating Constraints" section of the "Using Timing Constraints" chapter.

–report (Generate Hierarchy Loss Report)

The –report option generates an ARF report file. This report contains information about original net and instance names that were lost during back-annotation. Determination of lost nets are based on a flat view of the design. If a net traverses hierarchical boundaries, a loss is reported only if the entire net is removed. This report's introduction explains the reasons for lost net and instance names and provides a key for how to read the report.

This option must be run with an NGM file present. If it is run without this file, NGDAnno issues a warning and does not produce the report file.

Note This option is not recommended for users of synthesis tools that have unpredictable naming conventions.

–s (Change Speed)

-s [*speed*]

The –s option instructs NGDAnno to annotate the device speed you specify to the NGA file. The device *speed* can be entered with or without the leading dash. For example, both –s 3 and –s –3 are allowable entries.

Some architectures support the –s min option. This option instructs NGDAnno to annotate a minimum delay, rather than a maximum

worst-case delay, to the NGA file. The command line syntax is the following.

```
-s min
```

–s –min is not an allowable entry.

Note Settings made with the –s min option override any prorated timing parameters in the PCF file.

Dedicated Global Signals in Back-Annotation Simulation

This section presents information on how global signals are treated in back-annotation simulation.

Note For a description of the STARTUP, STARTUP_VIRTEX, STARTUP_VIRTEX2, and STARTUP_SPARTAN2 components, see the "Design Elements (SOP3 to XORCY_L)" chapter of the *Libraries Guide*.

XC3000A/L and 3100A/L

In XC3000 and XC3100 devices, the global reset signal (whose name varies depending on the CAE vendor) is assigned a pin on the device. You must include this pin in your call to the top level module and stimulate the pin. The global reset signal should be pulsed Low to reset all flip-flops in the design, then held High for normal operation.

XC4000E/L/EX/XL/XV/XLA and Spartan/XL

For the XC4000 device family, Spartan devices, and SpartanXL devices, a High signal on the GSR (Global Set/Reset) net initializes each flip-flop and latch to the state (0 or 1) specified by its INIT property (default is 0). For XC4000 devices, the INIT property must match the flip-flop type, except in the case of FD and LD components. A High signal on GTS (Global Tristate) sets all outputs to a tristate condition. If you have not used the STARTUP component in your original design, these signals are initialized to their inactive states. Otherwise, you must stimulate the input GSR and GTS pins of the STARTUP device either directly or through logic from explicit pins on the device.

XC5200

In XC5200 devices, GR (Global Reset) is assigned a pin on the device. You must include this pin in your call to the top level module and stimulate the pin. The global reset signal is active-High. A High signal on GTS (Global Tristate) sets all outputs to a tristate condition. If you have not used the STARTUP component in your original design, these signals are initialized to their inactive states. Otherwise, you must stimulate the input GR and GTS pins of the STARTUP device either directly or through logic from explicit pins on the device.

Virtex/-II/-E and Spartan-II

For Virtex, Virtex-II, Virtex-E, and Spartan-II devices, a High signal on the GSR (Global Set/Reset) net initializes each flip-flop and latch to the state (0 or 1) specified by its INIT property (default is 0) and each Block RAM data output to 0. The INIT property *must* match the flip-flop type. For example, if you use an FDR flip-flop, you must retain the automatically assigned INIT=R property. The DLL and the contents of the following memory elements are unaffected by GSR: LUT RAM, Block RAM, SRL16, and SRLC16. A High signal on GTS (Global Tristate) sets all outputs to a tristate condition. If you have not used the STARTUP_VIRTEX component in your original design, these signals are initialized to their inactive states. Otherwise, you must stimulate the input GSR and GTS pins of the STARTUP_VIRTEX device either directly or through logic from explicit pins on the device.

Virtex-II devices differ slightly from Virtex devices. The startup block for Virtex-II is called STARTUP_VIRTEX2. In addition, GSR has two levels of control. By default when GSR is asserted, a register sets or presets according to its type. For example, an FDR flip-flop changes to 0 when GSR is asserted. With Virtex-II, you can also override this default behavior by using the INIT property. For example, if you assign INIT=S for FDR, asserting GSR changes the state of the register to 1 instead of the default value of 0.

Note Using a BUFGMUX element in your design may cause inaccurate simulation if all the following conditions occur: both clock inputs (I0 and I1) are used, GSR is activated during simulation (after simulation time '0'), and the secondary clock input (I1) is selected before or while GSR is active. In this case, the primary clock input (I0) is incorrectly selected. This occurs because there is a cross-coupled register pair that ensures the BUFGMUX output does not inadvertently generate a clock edge. When GSR is asserted, these registers initialize to the default state of I0. To select the secondary clock, you must send a clock pulse to both the primary and secondary clock inputs while GSR is inactive.

Hierarchy Changes in Annotated Designs

If you supply an NGM file as input, NGDAnno attempts to retain your original design hierarchy. However, NGDAnno may flatten part of your original design hierarchy when generating a simulation netlist under the following conditions.

- Logical correlation loss on a CLB, IOB, or slice due to logic optimization and logic replication during mapping
- Failures in logical annotation
- Lost logic is located in different parts of the design hierarchy

For example, if a flip-flop with the hierarchical name A/B/X is merged with a flip-flop named A/C/Y, and the resulting CLB is affected by optimization or logic replication, hierarchical blocks A/B and A/C are flattened out of the netlist. The two flip-flops now lie at the same level of hierarchy (the A level) and are replaced by the CLB physical model.

For information on which blocks were flattened, see the ARF file generated when you use the –report option described in the "–report (Generate Hierarchy Loss Report)" section.

Guaranteed Setup and Hold Check

In addition to the setup and hold checks already performed during back-annotation, NGDAnno creates a Guaranteed Setup and Hold Check (GSUH) primitive between each input pad that drives an IOB register and global clock pad that drives the register's clock pin. The GSUH values match closely with the pin-to-pin setup and hold

values published in *The Programmable Logic Data Book*. Only device families and configurations that have published pin-to-pin setup and hold values are supported by the new GSUH primitive.

This checking only occurs for IOBs and clock configurations that meet the requirements for a particular device. Generally, devices require an externally driven clock that uses the global clock resources to clock an input IOB register. Older device families may not support all GSUH combinations.

Note Clock and data signals must be routed to the same hierarchical level for the GSUH primitive to be created. If a GSUH primitive cannot be created, NGDAnno issues a warning.

Chapter 19

NGD2EDIF

This program is compatible with the following families.

- Spartan / XL / -II
- Virtex / -E / -II
- XC9500 / XL / XV
- XC4000E / L / EX / XL / XV / XLA
- XC3000A / L
- XC3100A / L
- XC5200

This chapter describes the NGD2EDIF program. The chapter contains the following sections.

- "NGD2EDIF"
- "NGD2EDIF Syntax"
- "NGD2EDIF Files"
- "NGD2EDIF Options"
- "XMM (RAM Initialization) File"
- "EDIF Identifier Naming Conventions"

NGD2EDIF

NGD2EDIF produces an EDIF 2 0 0 netlist in terms of the Xilinx primitive set, allowing you to simulate pre- and post-route designs.

NGD2EDIF can produce an EDIF file representing a design in any of these stages.

- An unmapped design—To translate an unmapped design, the input to NGD2EDIF is an NGD file—a logical description of your design. The output from NGD2EDIF is an EDIF file containing a functional description of the design without timing information.
- A mapped, unrouted design—To translate a mapped design that has not been placed and routed, the input to NGD2EDIF is an NGA file— an annotated logical description of your design—generated from a mapped physical design. The output from NGD2EDIF is an EDIF file containing a functional description of the design and timing information containing component delays but without routing delays.
- A routed design—To translate a design which has been placed and routed, the input to NGD2EDIF is an NGA file generated from a routed physical design. The output from NGD2EDIF is an EDIF file containing a functional description of the design and timing information containing both component and routing delays.

The design flow for NGD2EDIF is shown in the following figure.

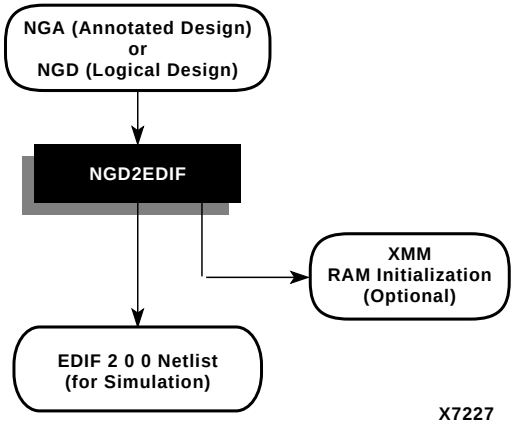


Figure 19-1 NGD2EDIF Design Flow

Note If you use a prohibited core in your design, NGD2EDIF issues an error message and does not export your design. If you use an encrypted core, NGD2EDIF generates an encrypted file.

NGD2EDIF Syntax

To invoke the NGD2EDIF translation program from the UNIX or DOS command line, enter the following.

ngd2edif [*options*] *infile*[**.ngd** | **.nga**] [*outfile*[**.edn**]]

options can be any number of the NGD2EDIF options listed in the "NGD2EDIF Options" section. They do not need to be listed in any particular order. Separate multiple options with spaces.

infile[**.ngd** | **.nga**] indicates the input file. If you enter a file name without an extension, NGD2EDIF looks for a file with an .nga extension and the name you specified. If you want to translate an NGD file, you must enter the .ngd extension. Without the .ngd extension NGD2EDIF does not use the NGD file as input, even if an NGA file is not present.

outfile[**.edn**] is the name of the NGD2EDIF output file if you want to name it other than the root NGD design name. If you do not give an extension, .edn is added.

Note If you are using the Viewlogic design entry tools, the *outfile* name must be different from the original design name to avoid conflict with the original WIR and EDIF files. See the "Timing Simulation" chapter in the *Viewlogic Interface Guide* for details.

NGD2EDIF Files

This section describes the NGD2EDIF input and output files.

Input Files

Input to the NGD2EDIF program can be either of the following files.

- NGA file—This back-annotated logical design file contains Xilinx primitive components.
- NGD file—This logical design file contains Xilinx primitive components.

Output Files

Output from NGD2EDIF consists of the following files.

- EDN file—This is a netlist file in EDIF format. The default EDN file produced by NGD2EDIF is generic. If you want to produce EDIF targeted to Mentor Graphics or Viewlogic, you must include the –v option (described in the "–v (Vendor)" section) on the command line.
- XMM file— This optional RAM initialization file defines the initial contents of the RAMs in the design for the simulator. The file is described in the "XMM (RAM Initialization) File" section.

  If an XMM file is generated, it has the same base name and is written into the same directory as the output EDIF netlist.

NGD2EDIF Options

This section describes the NGD2EDIF command options.

–a (Write All Properties)

The –a option causes NGD2EDIF to write all properties into the output EDIF netlist. The default is to write only timing delay properties and certain other properties that define the behavior of the design logic (for example, RAM INIT properties). In most cases the –a option is not necessary. Check with your simulation vendor on whether this option is a requirement for their tools.

–b (Use Buffers to Model Delays)

The –b option causes NGD2EDIF to model certain delays using buffers. The proper setting for the –b and –i options is chosen automatically if you entered a –v option. If your SimPrim library vendor is not one of the supported values for the –v option, consult the vendor for the proper –b and –i option settings.

–c (Reference Design Name as Specified—Mentor)

The –c option applies to the Mentor Graphics design flow. The option ensures that the output of Mentor's ENRead (EDIF reader) program is an EDDM Single Object simulation model registered to the design component located in the current directory.

If the –c option is *not* specified, a library entry in the EDIF file instructs ENRead to place the simulation model in a subdirectory named *design*_lib. For example, if your design name is adder4, ENRead places the simulation model in the subdirectory adder4_lib/adder4.

If the –c option *is* specified, the library entry in the EDIF file instructs ENRead to place the simulation model directly in the design directory. For example, the simulation model for the design adder4 is placed in the current directory right under adder4 (as opposed to being placed in adder4_lib/adder4). In this directory, the Mentor simulator finds the simulation model.

If you specify the –c option, you must also specify both the –n (Generate Flattened Netlist) option and the –v (Vendor) option, with the –v option specifying **–v mentor**.

–f (Execute Commands File)

-f *command_file*

The –f option executes the command line arguments in the specified *command_file*. For more information on the –f option, see the "–f Option" section of the "Introduction" chapter

–hpn (Set HDL Pin Names)

The –hpn option sets SimPrim pin names to HDL compliant names. The pin name IN is set to I, and the pin name OUT is set to O.

–i (Annotate Timing Properties to Instances)

The –i option causes NGD2EDIF to annotate all timing properties to instances. The proper setting for the –i and –b options are chosen automatically if you entered a –v option. If your SimPrim library vendor is not one of the supported values for the –v option, consult the vendor for the proper –i and –b option settings.

–l (Local Scope)

The –l option gives dedicated signals (such as the Global Set/Reset signal) local (non-global) scope. If your simulation vendor is Foundation, Mentor Graphics, or Viewlogic, the default NGD2EDIF action is to give dedicated signals global scope. If you are simulating

a board-level schematic that references more than one Xilinx device, the global dedicated signals from each netlist are implicitly connected by the simulator. If this is not what you want, use the –l option to make the signals local to each device. You then need to reference each dedicated signal by the appropriate hierarchically-qualified signal name.

If your simulation vendor is not Foundation, Mentor Graphics, or Viewlogic, the –l option is enabled by default.

–n (Generate Flattened Netlist)

The –n option writes out a flattened netlist.

–v (Vendor)

-v *vendor*

The –v option specifies the CAE vendor toolset that uses the resulting EDIF file. Allowable entries are **viewlog** (for Viewlogic), **mentor**, and **fndtn** (for Foundation).

The –v option customizes the output EDIF file for the specified vendor's simulator. The option also determines whether an XMM (RAM initialization) file is produced and the format of the file (if one is produced). The XMM file is described in the "XMM (RAM Initialization) File" section.

–vpt (Mentor Viewpoint)

-vpt *viewpoint_name*

The –vpt option specifies the desired viewpoint for a Mentor EDIF output.

–w (Overwrite Output)

The –w option overwrites the output file.

XMM (RAM Initialization) File

The XMM file defines the initial contents of the RAMs in the design for the simulator. An XMM file is only created if the design contains RAMs. Some simulators require an XMM file; other simulators can read the RAM initialization directly from the output EDIF file and do

not need a separate XMM file. The way you use the file depends on the simulator vendor you specify with the –v option to NGD2EDIF.

- If you are using a Viewlogic simulator (–v viewlog), an XMM file is created in LOADM format for use by ViewSim. See the "Timing Simulation" chapter in the *Viewlogic Interface Guide* for information on loading the XMM file into ViewSim.
- If you are using a Mentor Graphics simulator (–v mentor), no XMM file is created. QuickSim takes the RAM initialization information directly from the EDIF netlist.
- If you are using another simulator (no –v option), an XMM file is generated in a generic format, which is described in the next section. Your simulator may or may not need this separate file; consult your vendor's documentation for details.

Note RAM initialization data is not created for the Virtex Block RAM.

Generic File Format for XMM File

This section describes the format of the generic XMM file, which is created when the –v option is not specified for NGD2EDIF. Consult you simulator vendor's documentation to determine how to use this generic XMM file.

In most cases you do not need to understand the format of the generic XMM file. The following information is provided for reference. For ease of processing by scripting languages, the generic initialization file consists of newline-separated records. Each record has the following three tokens, separated by white space, with the position of each token denoting its meaning.

```
primitive_type instance_name init_value
```

The tokens are defined as follows.

primitive_type is the name of a RAM primitive in the SimPrim library. It is a string value.

instance_name is a hierarchically-qualified instance name for a particular RAM SimPrim in the design. It is a string value. Hierarchical names are separated by the forward slash (/) character. The *instance_name* is expressed in terms of the names in the original design, not in terms of the EDIF identifiers. The original names are more likely to correlate to the original design, but are not checked for

uniqueness and may not be legal for the simulation interface. The simulation interface must read the generic initialization file to resolve these problems.

init_value represents the contents of the specified RAM instance. The *init_value* is a hexadecimal number with a 0x prefix. The most significant bit of this number should be loaded into the highest address of the RAM, continuing so that the least-significant bit is loaded into the lowest address of the RAM. As with the INIT property value, a one-bit-wide RAM is assumed. The number is padded with zeroes so that the number of bits exactly matches the number of addressable locations in the RAM primitive.

Generic Initialization File Example

An example generic initialization file is shown following.

```
X_RAMS16 $1I32/$1I47/FIFO/BANK03 0x6A47
X_RAM32 TOP/IFC/DATA/O7 0x003F097D
X_RAMD16 TOP/$3I107/$7I100 0x0000
```

The generic initialization file also contains several comment lines that document when and how the file was made and describe the file format. Each comment line begins with a pound (#) character; these lines can be ignored by programs using the initialization file.

EDIF Identifier Naming Conventions

An identifier in a EDIF file must adhere to the following conventions.

- Must begin with an alphabetic or ampersand character (a–z, A–Z, or &)
- Can contain alphanumeric (a–z, A–Z, 0-9) or underscore (_) characters

Chapter 20

NGD2VER

This program is compatible with the following families.

- Spartan / XL / -II
- Virtex / -E / -II
- XC9500 / XL / XV
- XC4000E / L / EX / XL / XV / XLA
- XC3000A / L
- XC3100A / L
- XC5200

This chapter describes the NGD2VER program. The chapter contains the following sections.

- "NGD2VER"
- "NGD2VER Syntax"
- "NGD2VER Files"
- "NGD2VER Options"
- "Setting Global Set / Reset, Tristate, and PRLD"
- "Test Fixture File"
- "Bus Order in Verilog Files"
- "Verilog Identifier Naming Conventions"
- "Compile Scripts for Verilog Libraries"

NGD2VER

NGD2VER translates your design into a Verilog HDL file containing a netlist description of your design in terms of Xilinx simulation primitives. You can use the Verilog file to perform a back-end simulation with a Verilog simulator.

Simulation is based on SimPrims, which create simulation models using basic simulation primitives. For example, because a primitive for the XC4000 dual-port RAM does not exist in the Verilog SimPrim library files, NGD2VER builds a simulation model for the dual-port RAM out of two 16x1 RAM SimPrim primitives.

NGD2VER can produce a Verilog file representing a design at any of the following stages.

- An unmapped design—To translate an unmapped design, the input to NGD2VER is an NGD file—a logical description of your design. The output from NGD2VER is a Verilog file containing a functional description of the design without timing information.
- A mapped, unrouted design—To translate a mapped design which has not been placed and routed, the input to NGD2VER is an NGA file— an annotated logical description of your design—generated from a mapped physical design. The output from NGD2VER is a Verilog file containing a functional description of the design, and an additional SDF (Standard Delay Format) file containing timing information. The SDF file contains component delays without routing delays.
- A routed design—To translate a design that has been placed and routed, the input to NGD2VER is an NGA file generated from a routed physical design. The output from NGD2VER is a Verilog file containing a functional description of the design and an SDF file containing both component and routing delays.

The design flow for NGD2VER is shown in the following figure.

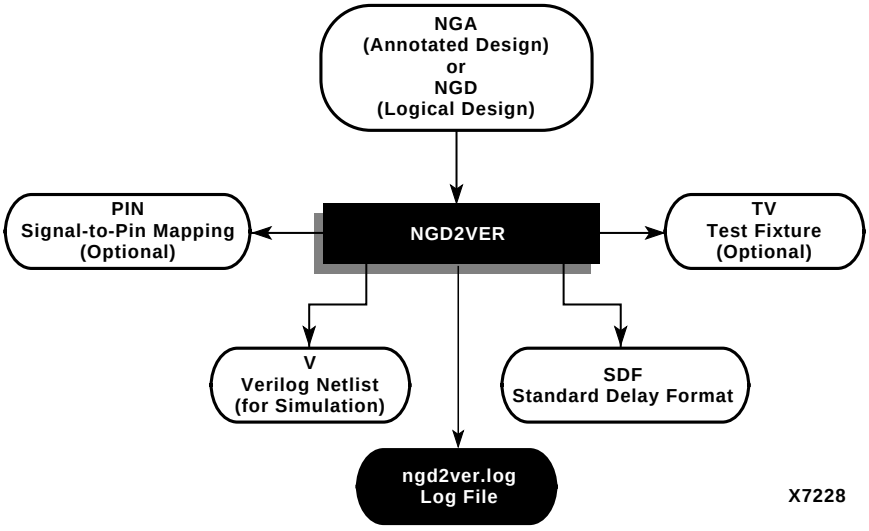


Figure 20-1 NGD2VER Design Flow

Note If you use a prohibited core in your design, NGD2VER issues an error message and does not export your design. If you use an encrypted core, NGD2VER generates an encrypted file.

NGD2VER Syntax

The following command translates your design to a Verilog file.

ngd2ver [*options*] *infile*[**.ngd** | **.nga**] [*outfile*[**.v**]]

options can be any number of the NGD2VER options listed in the "NGD2VER Options" section. They do not need to be listed in any particular order. Separate multiple options with spaces.

infile [**.ngd** | **.nga**] is the input NGD or NGA file. If you enter a file name with no extension, NGD2VER looks for a file with an .nga extension and the name you specified. If you want to translate an NGD file, you must enter the .ngd extension. Without the .ngd extension NGD2VER does not use the NGD file as input, even if an NGA file is not present.

outfile[.v] indicates the file to which the Verilog output of NGD2VER is written. The default is *infile*.v (*infile* is the same root name as the input file). The SDF file has the same root name as the Verilog file.

NGD2VER Files

This section describes the NGD2VER input and output files.

Input Files

Input to NGD2VER can be either of the following files.

- NGA—This back-annotated logical design file is produced by NGDAnno and contains Xilinx primitives.
- NGD—This logical design file is produced by NGDBuild and contains Xilinx simulation primitives.

Output Files

Output from NGD2VER consists of the following files.

- V file—This Verilog HDL file contains the netlist information obtained from the input NGD or NGA file. This file is a simulation model and cannot be synthesized or used in any other manner than simulation. This netlist uses simulation primitives which may not represent the true implementation of the device; however, the netlist represents a functional model of the implemented design. Do not modify this file.
- SDF file—This SDF (Standard Delay Format) file contains delays obtained from the input file. NGD2VER only generates an SDF file if the input is an NGA file, which contains timing information. The SDF file generated by NGD2VER is based on SDF version 2.1.

  Note The SDF file should only be used with the Verilog file. Do not use the SDF file with the original design or with the product of another netlist writer.
- LOG file—This log file contains all the messages generated during the execution of NGD2VER.
- TV file—This optional test fixture file is created if you enter the –tf option on the NGD2VER command line.

- PIN file—This is an optional Cadence signal-to-pin mapping file. NGD2VER generates a PIN file if the input file contains routed external pins and you have specified a –pf command line option.

  NGD2VER only generates an PIN file if the input is an NGA file. The files have the same root name as the NGA file.

NGD2VER Options

This section describes NGD2VER command options.

–10ps (Set Time Precision to be 10ps)

The –10ps option changes the default timescale statement from 1 ns / 1 ps to 1 ns / 10 ps. This allows you to choose the appropriate simulation resolution based on your simulation run-time requirements.

–aka (Write Also-Known-As Names as Comments)

The –aka option includes original user-defined identifiers as comments in the Verilog netlist. This option is useful if user-defined identifiers are changed because of name legalization processes in NGD2VER.

–cd (Include `celldefine\`endcelldefine in Verilog File)

The –cd option applies to a Verilog file that will be used with the Cadence Synergy synthesis tool. The –cd option encloses every module definition in `celldefine and `endcelldefine constructs, as shown below.

```
‘celldefine
  module <module_name>
       .
       .
       .
  endmodule
‘endcelldefine
```

The `celldefine and `endcelldefine constructs tell the Cadence Synergy software to treat an enclosed module as a black box (that is, do not try to synthesize the enclosed module).

This option is used if a Cadence Synergy user instantiates a Logi-BLOX module into the HDL source code.

–f (Execute Commands File)

-f *command_file*

The –f option executes the command line arguments in the specified *command_file*. For more information on the –f option, see the "–f Option" section of the "Introduction" chapter.

–gp (Bring Out Global Reset Net as Port)

-gp *port_name*

The –gp option causes NGD2VER to bring out the Global Reset signal (which is connected to all flip-flops and latches in the physical design) as a port on the top-level module in the output Verilog file. Specifying the port name allows you to match the port name you used in the front-end. The Global Reset signal is discussed in the "Dedicated Global Signals in Back-Annotation Simulation" section of the "NGDAnno" chapter.

This option is only used if the Global Reset net is not driven. For example, if you include a STARTUP component in an XC4000 design, you do not have to enter a –gp option, because the STARTUP component drives the Global Reset net.

Note Do not use GR, GSR, PRELOAD, or RESET as port names, because these are reserved names in the Xilinx software. Also, do not use the name of any wire or port that already exists in the design, because this causes NGD2VER to issue a fatal error.

–ism (Include SimPrim Modules in Verilog File)

The –ism switch includes SimPrim modules from the SimPrim library in the output Verilog (.v) file. This option allows you to bypass specifying the library path during simulation. However, using this switch increases the size of your netlist file and increases your compile time.

When you run this option, NGD2VER checks that your library path is set up properly. Following is an example of the appropriate path.

```
$XILINX/verilog/src/simprim
```

Note If you are using compiled libraries, this switch offers no advantage. If you use this switch, do not use the –ul switch.

–log (Rename the Log File)

-log *log_file*

By default, the name of the NGD2VER log file is ngd2ver.log. The –log option allows you to rename the log file. The log file contains all of the messages displayed during the execution of NGD2VER.

–ne (No Name Escaping)

By default (with no –ne option), NGD2VER "escapes" illegal block or net names in your design by placing a leading backslash (\) before the name and appending a space at the end of the name. For example, the net name "p1$40/empty" becomes "\p1$40/empty " when name escaping is used. Illegal Verilog characters are reserved Verilog names, such as "input" and "output," and any characters that do not conform to the standards described in the "Verilog Identifier Naming Conventions" section.

The –ne option replaces invalid characters with underscores so that name escaping does not occur. For example, the net name "p1$40/empty" becomes "p1$40_empty" when name escaping is not used. The leading backslash does not appear as part of the identifier. The resulting Verilog file can be used if a vendor's Verilog software cannot interpret escaped identifiers correctly.

–op (Specify the Period for Oscillator)

-op *oscillator_period*

The –op option specifies the period, in nanoseconds, for the oscillator. You must specify a positive integer to stimulate the component properly. If you do not enter a value for the –op option, the default is 100 ns.

Note Use this option only if OSC4 or OSC5 is included in your design.

–pf (Generate Pin File)

The –pf option writes out a pin file—a Cadence signal-to-pin mapping file with a .pin extension.

Note NGD2VER only generates an PIN file if the input is an NGA file.

–pms (Port Names Match Child Signal Names)

The –pms option forces port names and child signal names to match. Ports or pins in the NGD database generally have two connections, one to the port or parent signal and one to the child signal. In most cases, these signal names are the same. If the names are not the same, you can use this option to change the child signal names to match the parent signal name.

–r (Retain Hierarchy)

The –r option writes out a Verilog HDL file that retains the hierarchy in the original design as much as possible. This option groups logic based on the original design hierarchy. To run NGD2VER with the –r option, you *must* have supplied an NGM file as input when you ran NGDAnno (see the "Input Files" section of the "NGDAnno" chapter). If you did not supply an NGM file, the NGA file produced is based on the NCD file, rather than the original design hierarchy.

The default setting (with no –r option) produces a flattened Verilog HDL file.

Note In some cases, it is not possible to preserve hierarchy. If NGDAnno cannot back-annotate timing delays, it inserts a physical model into the logical model. If the logical elements for any CLB spanned hierarchy, the hierarchy is flattened as a result of this insertion. See the "Hierarchy Changes in Annotated Designs" section of the "NGDAnno" chapter for more information.

–sdf_path (Full Path to SDF File)

`-sdf_path` [*`path_name`*]

The –sdf_path option outputs the SDF file to the specified full path. This option writes the full path and the SDF file name to the $sdf_annotate statement. If a full path is not specified, it writes the full path of the current work directory and the SDF file name to the $sdf_annotate file.

Note NGD2VER only generates an SDF file if the input is an NGA file, which contains timing information. This option is allowed on an NGA file but not on an NGD file.

–shm (Write $shm Statements in Test Fixture File)

The -shm option places $shm statements in the structural Verilog file created by NGD2VER. These $shm statements allow VerilogXL to display simulation data as waveforms.

–tf (Generate Test Fixture File)

The –tf option generates a test fixture file. The file has a .tv extension, and it is a ready-to-use template test fixture Verilog file based on the input NGD or NGA file.

–ti (Top Instance Name)

-ti *top_instance_name*

The –ti option specifies a user instance name for the design under test in the test fixture file created with the -tf option.

–tm (Top Module Name)

-tm *top_module_name*

The –tm option changes the name of the top-level module name appearing within the NGD2VER output files. By default (with no –tm option), the output files inherit the top module name from the input NGD or NGA file.

–tp (Bring Out Global Tristate Net as Port)

-tp *port_name*

The –tp option causes NGD2VER to bring out the global tristate signal (which forces all FPGA outputs to the high-impedance state) as a port on the top-level entity in the output Verilog file. Specifying the port name allows you to match the port name you used in the front-end.

This option is only used if the global tristate net is not driven. For example, if you include a STARTUP component in an XC4000 design,

you do not have to enter a –tp option, because the STARTUP component drives the global tristate net.

Note Do not use the name of any wire or port that already exists in the design, because this causes NGD2VER to issue a fatal error.

–ul (Write ‘uselib Directive)

The –ul option causes NGD2VER to write a library path pointing to the SimPrim library into the output Verilog (.v) file. The path is written as shown following.

```
‘uselib dir=$XILINX/verilog/src/simprims libext=.v
```

$XILINX is the location of the Xilinx software.

If you do not enter a –ul option, the ‘uselib line is not written into the Verilog file.

Note A blank ‘uselib statement is automatically appended to the end of the Verilog file to clear out the ‘uselib data. If you use this switch, do not use the –ism switch.

–verbose (Display Processing Messages in Verbose Mode)

The –verbose option displays detailed Verilog processing messages during the execution of NGD2VER.

–w (Overwrite Existing Files)

The –w option causes NGD2VER to overwrite the .v file if it already exists. By default, NGD2VER does *not* overwrite the .v file.

Note All other output files are automatically overwritten.

Setting Global Set/Reset, Tristate, and PRLD

For information on setting Global Set/Reset and Global Tristate for FPGAs, see the “Defining Global Signals in Verilog” section of the *Synthesis and Simulation Design Guide.*

For information on setting Global PRLD for CPLDs, refer to the “Simulating Your Design” chapter of the *CPLD Synthesis Design Guide*.

Test Fixture File

The end of the test fixture (TV) file produced by NGD2VER contains the following commands.

```
#1000 $stop
// #1000 $finish
```

The `$stop` command terminates simulation from the test fixture and places the simulator in "interactive mode." This mode allows you to view the waveforms produced or allows interaction with other programs that need the simulator open.

To exit automatically instead of entering interactive mode, edit the test fixture file to remove or comment out the $stop line and uncomment the $finish line.

Bus Order in Verilog Files

When you compile your unit-under-test design from NGD2VER along with your test fixture, there may be mismatches on bused ports.

This problem occurs when your unit under test has top-level ports that are defined as LSB-to-MSB, as shown in the following example.

```
input [0:7] A;
```

As a result of the way your input design was converted to a netlist before it was read into the Xilinx implementation software, the Xilinx design database does not include information on how bus direction was defined in the original design. When NGD2VER writes out a structural timing Verilog description, all buses are written as MSB-to-LSB, as shown in the following example.

```
input [7:0] A;
```

If your ports are defined as LSB-to-MSB in your original input design and test fixture, there is a port mismatch when the test fixture is compiled for timing simulation. Use one of the following methods to solve this problem.

- In the test fixture, modify the instantiation of the unit under test so that all ports are defined as MSB-to-LSB for timing simulation.
- Define all ports as MSB-to-LSB in your original design and test fixture. For example, enter **[7:0]** instead of **[0:7]**.

Note Bus order *will* be preserved in the following cases: if the design input file is EDIF and the buses are declared as port arrays, if you are doing logical simulation, or if you are doing back-annotation with an NGM file as input.

Verilog Identifier Naming Conventions

An identifier in a Verilog file must adhere to the following conventions. For more information see the *IEEE Standard Description Language Based on the Verilog™ Hardware Description Language* manual.

- Must begin with an alphabetic or underscore character (a-z, A-Z, or _)
- Can contain alphanumeric (a-z, A-Z, 0-9), underscore (_), or dollar sign ($) characters
- May use any character by escaping with a backslash(\) at the beginning of the identifier and terminating with a white space (a blank, tab, newline, or formfeed). For example, the identifier "reset*" is not acceptable but the identifier "\reset* " is acceptable.
- Can be up to 1024 characters long
- Cannot contain white space

Note Identifiers are case sensitive.

During the name legalization process, NGD2VER writes identifiers that contain invalid characters with a leading backslash and a following white space. If you want to change this default behavior, use the –ne option described in the "–ne (No Name Escaping)" section.

Compile Scripts for Verilog Libraries

You must compile libraries for your simulation tools to recognize Xilinx components. To perform timing or post-synthesis functional HDL simulation, you must compile the SimPrim libraries. If the HDL code contains instantiated components, you must compile the UniSim or LogiBLOX libraries. If the HDL code contains instantiated components from the CORE Generator System, you must compile the CORE Generator behavioral models before you can perform a behavioral simulation. Refer to the *CORE Generator Guide* for more information.

To compile libraries, refer to the “Compiling HDL Libraries” section of the *Synthesis and Simulation Design Guide*.

Note You do not need to compile libraries for Verilog-XL.

Chapter 21

NGD2VHDL

This program is compatible with the following families.

- Spartan / XL / -II
- Virtex / -E / -II
- XC9500 / XL / XV
- XC4000E / L / EX / XL / XV / XLA
- XC3000A / L
- XC3100A / L
- XC5200

This chapter describes the NGD2VHDL program. The chapter contains the following sections.

- "NGD2VHDL"
- "NGD2VHDL Syntax"
- "NGD2VHDL Files"
- "NGD2VHDL Options"
- "VHDL Global Set / Reset Emulation"
- "Bus Order in VHDL Files"
- "VHDL Identifier Naming Conventions"
- "Compile Scripts for VHDL Libraries"

NGD2VHDL

The NGD2VHDL program translates your design into a VHDL file containing a netlist description of the design in terms of Xilinx simulation primitives. You can use the VHDL file to perform a back-end simulation by a VHDL simulator.

Simulation is based on SimPrims, which create simulation models using basic simulation primitives. For example, because a primitive for the XC4000 dual-port RAM does not exist in the VITAL SimPrim library files, NGD2VHDL builds a simulation model for the dual port ram out of two 16x1 RAM SimPrim primitives.

NGD2VHDL produces a VHDL file representing a design in any of the following stages.

- An unmapped design—To translate an unmapped design, the input to NGD2VHDL is an NGD file—a logical description of your design. The output from NGD2VHDL is a VHDL file containing a functional description of the design without timing information.
- A mapped, unrouted design—To translate a mapped design which has not been placed and routed, the input to NGD2VHDL is an NGA file— an annotated logical description of your design—generated from a mapped physical design. The output from NGD2VHDL is a VHDL file containing a functional description of the design, and an additional SDF (Standard Delay Format) file containing timing information. The SDF file contains component delays without routing delays.
- A routed design—To translate a design which has been placed and routed, the input to NGD2VHDL is an NGA file generated from a routed physical design. The output from NGD2VHDL is a VHDL file containing a functional description of the design and an SDF file containing both component and routing delays.

The design flow for NGD2VHDL is shown in the following figure.

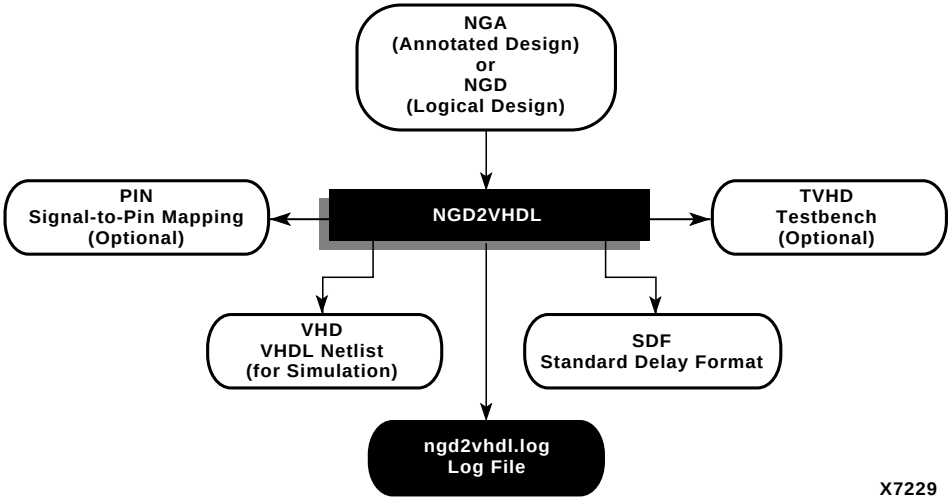


Figure 21-1 NGD2VHDL Design Flow

Note If you use a prohibited core in your design, NGD2VHDL issues an error message and does not export your design. If you use an encrypted core, NGD2VHDL generates an encrypted file.

NGD2VHDL Syntax

The following command translates your design to a VHDL file.

ngd2vhdl [*options*] *infile*[**.ngd** | **.nga**] [*outfile*[**.vhd**]]

options can be any number of the NGD2VHDL options listed in the "NGD2VHDL Options" section. They do not need to be listed in any particular order. Separate multiple options with spaces.

infile [**.ngd** | **.nga**] is the input NGD or NGA file. If you enter a file name without an extension, NGD2VHDL looks for a file with an .nga extension and the name you specified. If you want to translate an NGD file, you must enter the .ngd extension. Without the .ngd extension NGD2VHDL does not use the NGD file as input, even if an NGA file is not present.

outfile[**.vhd**] indicates the file to which the VHDL output of NGD2VHDL is written. The default is *infile*.vhd (*infile* is the same root name as the input file). The SDF file has the same root name as the VHDL file.

NGD2VHDL Files

This section describes the NGD2VHDL input and output files.

Input Files

Input to NGD2VHDL can be any of the following files.

- NGA—This back-annotated logical design file contains Xilinx primitive components.
- NGD—This logical design file contains Xilinx primitive components.

Output Files

Output from NGD2VHDL consists of the following files.

- VHD file—This VITAL 95 IEEE compliant VHDL file contains the netlist information obtained from the input NGD or NGA file. This file is a simulation model and cannot be synthesized or used in any other manner than simulation. This netlist uses simulation primitives which may not represent the true implementation of the device; however, the netlist represents a functional model of the implemented design. Do not modify this file.
- SDF file—This Standard Delay Format file contains delays obtained from the input file. NGD2VHDL only generates an SDF file if the input is an NGA file, which contains timing information. The SDF file generated by NGD2VHDL is based on SDF version 2.1.
- LOG file—This log file contains all the messages generated during the execution of NGD2VHDL.
- Testbench file—This optional testbench file is created if you enter the –tb option on the NGD2VHDL command line. The file has a .tvhd extension.

NGD2VHDL Options

This section describes the NGD2VHDL command options.

–a (Architecture Only)

By default, NGD2VHDL generates both entities and architectures for the input design. If the –a option is specified, no entities are generated and only architectures appear in the output.

–aka (Write Also-Known-As Names as Comments)

The –aka option includes original user-defined identifiers as comments in the VHDL netlist. This option is useful if user-defined identifiers are changed because of name legalization processes in NGD2VHDL.

–ar (Rename Architecture Name)

-ar *architecture_name*

The –ar option allows you to rename the architecture name generated by NGD2VHDL. The default architecture name for each entity in the netlist is STRUCTURE.

–f (Execute Commands File)

-f *command_file*

The –f option executes the command line arguments in the specified *command_file*. For more information on the –f option, see the "–f Option" section of the "Introduction" chapter.

–gp (Bring Out Global Reset Net as Port)

-gp *port_name*

The –gp option causes NGD2VHDL to bring out the Global Reset signal (which is connected to all flip-flops and latches in the physical design) as a port on the top-level entity in the output VHDL file. Specifying the port name allows you to match the port name you used in the front-end. The Global Reset signal is discussed in the "VHDL Global Set/Reset Emulation" section.

This option is only used if the Global Reset net is not driven. For example, if you include a STARTUP component in an XC4000 design, you do not have to enter a –gp option, because the STARTUP component drives the Global Reset net.

Note Do not use GR, GSR, PRELOAD, or RESET as port names, because these are reserved names in the Xilinx software.

–log (Specify the Log File)

-log *log_file*

By default, the name of the NGD2VHDL log file is ngd2vhdl.log. The –log option allows you to rename the log file. The log file contains all of the messages displayed during the execution of NGD2VHDL.

–op (Specify the Period for Oscillator)

-op *oscillator_period*

The –op option specifies the period, in nanoseconds, for the oscillator. You must specify a positive integer to stimulate the component properly. If you do not enter a value for the –op option, the default is 100 ns.

Note Use this option only if OSC4 or OSC5 is included in your design.

–pms (Port Names Match Child Signal Names)

The –pms option forces port names and child signal names to match. Ports or pins in the NGD database generally have two connections, one to the port or parent signal and one to the child signal. In most cases, these signal names are the same. If the names are not the same, you can use this option to change the child signal names to match the parent signal name.

–r (Retain Hierarchy)

The –r option writes out a VHDL file that retains the hierarchy in the original design as much as possible. This option groups logic based on the original design hierarchy. To run NGD2VHDL with the –r option, you *must* have supplied an NGM file as input when you ran NGDAnno (see the “Input Files” section of the “NGDAnno” chapter).

If you did not supply an NGM file, the NGA file produced is based on the NCD file, rather than the original design hierarchy.

The default setting (with no –r option) produces a flattened VHDL file.

Note In some cases, it is not possible to preserve hierarchy. If NGDAnno cannot back-annotate timing delays, it inserts a physical model into the logical model. If the logical elements for any CLB spanned hierarchy, the hierarchy is flattened as a result of this insertion. See the "Hierarchy Changes in Annotated Designs" section of the "NGDAnno" chapter for more information.

–rpw (Specify the Pulse Width for ROC)

-rpw *roc_pulse_width*

The –rpw option specifies the pulse width, in nanoseconds, for the ROC component. You must specify a positive integer to stimulate the component properly. This option is not required. By default, the ROC pulse width is set to 100 ns.

–tb (Generate Testbench File)

The –tb option writes out a testbench file with a .tvhd extension. The default top-level instance name within the testbench file is UUT. If you enter a –ti (Top Instance Name) option, the top-level instance name is the name specified by the –ti option.

–te (Top Entity Name)

-te *top_entity_name*

The –te option specifies the name of the top-level entity in the structural VHDL file produced by NGD2VHDL for timing simulation. By default (with no –te option), the output files inherit the top entity name from the input NGD or NGA file.

–ti (Top Instance Name)

-ti *top_instance_name*

The –ti option specifies the name of the top-level instance name appearing within the output SDF file and testbench file (if produced).

The option allows you to match the top-level instance name to the name specified in your test driver VHDL file. Without this option, the SDF file generated by NGD2VHDL cannot be processed properly by VHDL simulators (for example, Model Technology vsim) for timing simulation.

If you do not enter a –ti option, the output files contain a top-level instance name of UUT.

–tp (Bring Out Global Tristate Net as Port)

-tp *port_name*

The –tp option causes NGD2VHDL to bring out the global tristate signal (which forces all FPGA outputs to the high-impedance state) as a port on the top-level entity in the output VHDL file. Specifying the port name allows you to match the port name you used in the front-end.

This option is only used if the global tristate net is not driven. For example, if you include a STARTUP component in an XC4000 design, you do not have to enter a –tp option, because the STARTUP component drives the global tristate net.

–tpw (Specify the Pulse Width for TOC)

-tpw *toc_pulse_width*

The –tpw option specifies the pulse width, in nanoseconds, for the TOC component. You must specify a positive integer to stimulate the component properly. This option is required when you instantiate the TOC component (for example, when the Global Set/Reset and Tristate nets are sourceless in the design).

–verbose (Display Processing Messages in Verbose Mode)

The –verbose option displays detailed VHDL processing messages when NGD2VHDL is run.

–w (Overwrite Existing Files)

The –w option causes NGD2VHDL to overwrite the .vhd file if it exists. By default, NGD2VHDL does *not* overwrite the .vhd file.

Note All other output files are automatically overwritten.

–xon (Select Output Behavior for Timing Violations)

```
-xon {true|false}
```

The –xon option specifies the output behavior when timing violations occur on memory elements. If you set this option to true, any memory elements that violate a setup time trigger X on the outputs. If you set this option to false, the signal's previous value is retained. If you do not set this option, –xon true is run.

VHDL Global Set/Reset Emulation

VHDL requires ports for all signals to be controlled by a testbench. There are VHDL specific components that can be instantiated in the RTL and post-synthesis VHDL description in order to enable the simulation of the global signals for Global Set/Reset and Global Tristate. NGD2VHDL creates a port on the back-annotated design entity for stimulating the Global Set/Reset or Tristate enable signals. This port does not actually exist on the configured part.

You do not need to use the –gp switch to create an external port if you instantiated a STARTUP block in the implemented design. In this case, the port is already identified and connected to the Global Set/Reset or Tristate enable signal. If you do not use the –gp option or a STARTUP block, you will need to use a special cell. Detailed directions for specific emulation cells and their uses follow.

Note The term "STARTUP" refers to the STARTUP block for all device families, including the Virtex STARTUP block, STARTUP_VIRTEX, and the Virtex-II STARTUP block, STARTUP_VIRTEX2. The term "STARTBUF" refers to the STARTBUF cell for all device families, including the Virtex STARTBUF cell, STARTBUF_VIRTEX, and the Virtex-II STARTBUF cell, STARTBUF_VIRTEX2.

VHDL Only STARTUP Block

The STARTUP block is traditionally instantiated to identify the GR, PRLD, or GSR signals for implementation. However, the only time simulation is enabled in the traditional method is when the net attached to the GSR or GTS also goes off chip, because the STARTUP block does not have simulation models. You can use the following

new cells to simulate global set / reset or tristate nets in all cases, whether or not the signal goes off chip.

Note The Virtex and Virtex-II STARTUP blocks are subsets of the XC4000 STARTUP block. However, they differ from the XC4000 STARTUP block in that they have no outputs, as shown in the following figure.

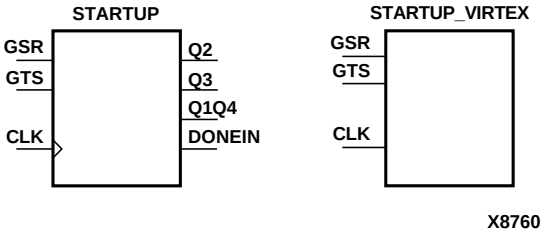


Figure 21-2 STARTUP and STARTUP_VIRTEX Blocks

VHDL Only STARTBUF Cell

The STARTBUF cell passes a Reset or Tristate signal in the same way that a buffer allows simulation to proceed, and it also instantiates the STARTUP block for implementation. STARTBUF is a more simulation friendly version of a typical STARTUP block. There is one version that works for all technologies, even though the XC5200 and the XC4000 STARTUP blocks have different pin names. Implementation with the correct STARTUP block is handled automatically. An instantiation example for the STARTBUF cell follows.

```
U1: STARTBUF port map (GSRIN => DEV_GSR_PORT, GTSIN
=>DEV_GTS_PORT, CLKIN => '0', GSROUT => GSR_NET,
GTSOUT => GTS_NET, Q2OUT => open, Q3OUT => open,
Q1Q4OUT => open, DONEINOUT => open):
```

One or both of the input ports GSRIN and GTSIN of the STARTBUF component and the associated output ports GSROUT and GTSOUT can be used. The pins that are left "open" can be used to pass configuration instructions down to implementation, just as on a traditional STARTUP block. You can do this by connecting the appropriate signal to the port instead of leaving it in an "open" condition.

Note The STARTBUF_VIRTEX and STARTBUF_VIRTEX2 cells are similar to the STARTBUF cell, but GSROUT is not available.

VHDL Only STARTUP_VIRTEX Block and STARTBUF_VIRTEX Cell

Global Set/Reset and Global Tristate for the Virtex STARTUP block, STARTUP_VIRTEX, and STARTBUF cell, STARTBUF_VIRTEX, operate as described in the preceding sections with the following qualifications.

Note This information also applies to STARTUP_VIRTEX2 and STARTBUF_VIRTEX2.

- Pre-NGDBuild UniSim VHDL simulation of the GSR signal is not supported.

  The simulation libraries will start up in the correct state; however, you cannot reset the design after simulation time '0.'

- During Pre-NGDBuild UniSim VHDL simulation, designs are properly initialized at simulation time '0.'

- Post-NGDBuild SimPrim VHDL simulation of GSR is supported.

  To correctly back-annotate a GSR signal, instantiate a STARTUP_VIRTEX or STARTBUF_VIRTEX symbol and correctly connect the GSR input signal of that component. When back-annotated, your GSR signal is correctly connected to the associated registers and RAM blocks.

- Pre-NGDBuild UniSim VHDL simulation of the GTS signal is supported.

  Instantiate either a STARTBUF_VIRTEX, TOC, or TOCBUF for this functionality.

VHDL Only RESET-ON-CONFIGURATION (ROC) Cell

This cell is created during back-annotation if you do not use the –gp option or STARTUP block options. It can be instantiated in the front end to match functionality with GSR, GR, or PRLD. (This is done in both functional and timing simulation.) During back-annotation, the entity and architecture for the ROC cell is placed in the design's output VHDL file. In the front end, the entity and architecture are in the UniSim Library, and require only a component instantiation.

The ROC cell generates a one-time initial pulse to drive the GR, GSR, or PRLD net starting at time '0' for a user-defined pulse width. You

can set the pulse width with a generic in a configuration statement. The default value of "width" is 0 ns, which disables the ROC cell and results in the Global Set/Reset being held Low. (Active-Low resets are handled within the netlist itself and require you to invert this signal before using it.)

The ROC cell allows you to simulate with the same testbench as in the RTL simulation, and also allows you to control the width of the global set/reset signal in the implemented design.

The ROC components require a value for the generic WIDTH, usually specified with a configuration statement. Otherwise, a generic map is required as part of the component instantiation.

You can set the generic with any generic mapping method you choose. Set the "width" generic after consulting *The Programmable Logic Data Book* for the particular part and mode you have implemented.

For example, a XC4000E part can vary from 10 ms to 130 ms. The value to look for is the TPOR (Power-ON Reset) parameter found in the Configuration Switching Characteristics tables for master, slave, and peripheral modes.

One of the easiest methods for mapping the generic is a configuration for the user's testbench. An example testbench configuration for setting the generic is as follows.

```
CONFIGURATION cfg_my_timing_testbench OF my_testbench IS
  FOR my_testbench_architecture
    FOR ALL:my_design USE ENTITY work.my_design(structure);
      FOR structure
        FOR ALL:roc ENTITY USE work.roc (roc_v)
          Generic MAP (width => 100 ms);
        END FOR;
      END FOR;
    END FOR;
  END FOR;
END cfg_my_timing_testbench;
```

The following is an instantiation example for the ROC cell.

```
U1: ROC port map (0 =>GSR_NET);
```

VHDL Only ROCBUF Cell

The ROCBUF allows you to provide stimulus for the Reset on Configuration signal through a testbench but the port connected to it is not implemented as a chip pin. The port can be brought back in the timing simulation with the –gp switch on NGD2VHDL. An example of instantiation of the ROCBUF cell follows.

```
U1: ROCBUF port map (I => SIM_GSR_PORT, O => GSR_NET);
```

Note This cell is not available for Virtex, Virtex-E, Virtex-II, and Spartan-II devices.

VHDL Only Tristate-On-Configuration (TOC) Cell

This cell is created if you do not use the –tp or StartUp block options. The entity and architecture for the TOC cell is placed in the design's output VHDL file. The TOC cell generates a one-time initial pulse to drive the GR, GSR, or PRLD net starting at time '0' for a user-defined pulse width. The pulse width can be set with a generic. The default value of "width" is 0 ns, which disables the TOC cell and results in the Tristate enable being held Low. (Active-Low Tristate enables are handled within the netlist itself and require you to invert this signal before using it.)

The TOC cell allows you to simulate with the same testbench as in the RTL simulation, and also allows you to control the width of the Tristate enable signal in the implemented design.

The TOC components require a value for the generic WIDTH, usually specified with a configuration statement. Otherwise, a generic map is required as part of the component instantiation.

You may set the generic with any generic mapping method you choose. Set the "width" generic after consulting *The Programmable Logic Data Book* for the particular part and mode you have implemented.

For example, a XC4000E part can vary from 10 ms to 130 ms. The value to look for is the TPOR (Power-ON Reset) parameter found in the Configuration Switching Characteristics tables for master, slave, and peripheral modes.

One of the easiest methods for mapping the generic is a configuration for the user's testbench. An example testbench configuration for setting the generic is as follows.

```
CONFIGURATION cfg_my_timing_testbench OF my_testbench IS

  FOR my_testbench_architecture
    FOR ALL:my_design USE ENTITY work.my_design(structrue);
      FOR structure
        FOR ALL:toc ENTITY USE work.toc (toc_v)
          Generic MAP (width => 100 ms);
        END FOR;
      END FOR;
    END FOR;
  END FOR;
END cfg_my_timing_testbench;
```

VHDL Only TOCBUF

The TOCBUF allows you to provide stimulus for the Global Tristate signal (GTS) through a testbench but the port connected to it is not implemented as a chip pin. The port can be brought back in the timing simulation with the –tp switch on NGD2VHDL. An example of the instantiation of the TOCBUF cell follows.

```
U2: TOCBUF port map (I =>SIM_GTS_PORT, O =>GTS_NET);
```

VHDL Only Oscillators

Oscillator output can vary within a fixed range. The cell is not included in the SimPrim library, because you cannot drive global signals in VHDL designs. Schematic simulators can define and drive global nets so the cell is not required. Verilog has the ability to drive nets within a lower level module as well. Therefore the oscillator cells are only required in VHDL. After back-annotation, their entity and architectures are contained in the design's VHDL output.

For functional simulation, they may be instantiated and simulated with the UniSim Library.

The period of the base frequency must be set in order for the simulation to proceed, because the default period of 0 ns disables the oscillator. The oscillator's frequency can very significantly with process and temperature.

Before you set the base period parameter, consult *The Programmable Logic Data Book* for the particular part you are using. For example, the section in *The Programmable Logic Data Book* for the XC4000 Series On-Chip Oscillator states that the base frequency can vary from 4 MHz to 10 MHz, and is nominally 8 MHz. This means the base period generic "period_8m" in the XC4000E OSC4 VHDL model can range from 250 ns to 100 ns. An example of this follows.

Note The OSC4 cell is only available for the XC4000 device family and for Spartan and SpartanXL devices.

Example 1: Oscillator VHDL

Following is an example of the XC4000E OSC4 VHDL model.

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

library UNISIM;
use UNISIM.all;

entity test1 is
port (DATAIN: in STD_LOGIC;
DATAOUT: out STD_LOGIC);
end test1;

architecture inside of test1 is

signal RST: STD_LOGIC;

component ROC
port(O: out STD_LOGIC);
end component;

component OSC4
port(F8M: out STD_LOGIC);
end component;

signal internalclock: STD_LOGIC;
begin

U0: ROC port map (RST);
```

```
U1: OSC4 port map (F8M=>internalclock);

process(internalclock)
begin
if (RST='1') then
DATAOUT <= '0';

elsif(internalclock'event and internalclock='1') then
DATAOUT <= DATAIN;

end if;

end process;

end inside;
```

Example 2: Oscillator Test Bench

Following is a second example.

```
library IEEE;

use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

library UNISIM;
use UNISIM.all;

entity test_oftest1 is end test_oftest1;

architecture inside of test_oftest1 is

component test1
port(DATAIN: in STD_LOGIC;
DATAOUT: out STD_LOGIC);
end component;

signal userdata, userout: STD_LOGIC;

begin

UUT: test1 port map(DATAIN=>userdata,DATAOUT=>userout);
```

```
myinput: process
begin
userdata <= '1';
wait for 299 ns;
userdata <= '0';wait for 501 ns;
end process;

end inside;

configuration overall of test_oftest1 is
for inside
      for UUT:test1
        for inside
              for U0:ROC use entity UNISIM.ROC(ROC_V)
              generic map (WIDTH=> 52 ns);
              end for;

              for U1:OSC4 use entity UNISIM.OSC4(OSC4_V)
              generic map (PERIOD_8M=> 25 ns);
              end for;
        end for;
      end for;
end for;
end overall;
```

This configuration is for pre-NGDBuild simulation. A similar configuration is used for post-NGDBuild simulation. The ROC, TOC, and OSC4 are mapped to the WORK library, and corresponding architecture names may be different. Review the .vhd file created by NGD2VHDL for the current entity and architecture names for post-NGDBuild simulation.

Bus Order in VHDL Files

When you compile your unit-under-test design from NGD2VHDL with your testbench, there may be mismatches on bused ports.

This problem occurs when your unit under test has top-level ports that are defined as LSB-to-MSB, as shown in the following example.

```
A: in STD_LOGIC_VECTOR (0 to 7);
```

As a result of the way your input design was converted to a netlist before it was read into the Xilinx implementation software, the Xilinx design database does not include information on how bus direction was defined in the original design. When NGD2VHDL writes out a structural timing VHDL description, all buses are written as MSB-to-LSB, as shown in the following example.

```
A: in STD_LOGIC_VECTOR (7 downto 0);
```

If your ports were defined as LSB-to-MSB in your original input design and testbench, there is a port mismatch when the testbench is compiled for timing simulation. Use one of the following to solve this problem.

- In the testbench, modify the instantiation of the unit under test so that all ports are defined as MSB-to-LSB for timing simulation
- Define all ports as MSB-to-LSB in the original design and testbench, by using the downto clause instead of the to clause to specify a bus range.

Note Bus order *will* be preserved in the following cases: if the design input file is EDIF and the buses are declared as port arrays, if you are doing logical simulation, or if you are doing back-annotation with an NGM file as input.

VHDL Identifier Naming Conventions

An identifier in a VHDL file must adhere to the following conventions. For more information see the *IEEE Standard VHDL Language Reference Manual or the IEEE Standard VITAL Application-Specific Integrated Circuit (ASIC) Modeling Specification*.

- Must begin with alphabetic characters (a–z or A–Z)
- Can contain alphanumeric (a–z, A–Z, 0-9) or underscore (_) characters
- Can be up to 1024 characters long
- Cannot contain white space

Note Identifiers are *not* case sensitive.

During the name legalization process, NGD2VHDL substitutes any illegal characters with the underscore (_) character.

Compile Scripts for VHDL Libraries

You must compile libraries for your simulation tools to recognize Xilinx components. To perform timing or post-synthesis functional HDL simulation, you must compile the SimPrim libraries. If the HDL code contains instantiated components, you must compile the UniSim or LogiBLOX libraries. If the HDL code contains instantiated components from the CORE Generator System, you must compile the COREGen behavioral models before you can perform a behavioral simulation. Refer to the *CORE Generator System User Guide* for more information.

To compile libraries, refer to the “Compiling HDL Libraries” section of the *Synthesis and Simulation Design Guide*.

Chapter 22

XFLOW

XFLOW is compatible with the following families.

- Spartan / XL / -II
- Virtex / -E / -II
- XC3000A / L
- XC3100A / L
- XC4000E / L
- XC4000EX / XL / XV / XLA
- XC5200
- XC9500
- XC9500XL

The chapter contains the following sections.

- “Overview”
- “XFLOW Syntax”
- “Running XFLOW”
- “Flow Types”
- “Option Files”
- “XFLOW Options”
- “Input Files”
- “Output Files”

Overview

The purpose of XFLOW is to provide a mechanism for users to encapsulate the Xilinx implementation or simulation flows within their own tools. These user tools might be simple scripts or they might be company frameworks.

XFLOW is a command line tool that allows you to run the full suite of Xilinx implementation and simulation flows. To run, XFLOW reads a design file, flow file, and option files as inputs. The flow file specifies the sequence of Xilinx tools to run on a design. The option file specifies the command line options for each of the tools listed in the flow file. Xilinx has provided six option files for implementation: three each for FPGAs and CPLDs and numerous option files for simulation. See the "Option Files" section for a list of these files. Xilinx has also provided three flow files to perform some basic implementation sequences. See the "Flow Files" section for a list.

If you run XFLOW on a design for the first time, XFLOW searches through the hierarchy and copies the corresponding flow and option files into your working directory. For later runs with the same command line options, XFLOW searches through your install hierarchy (or tree) and copies the flow files and option files. The hierarchical search is as follows:

- Current working directory (First)
- Directories specified with the XFLOWPATH environment variable. This variable is used to define implementations for project-based, team-based, or company-wide settings. For example, consider a project team working on a design. If a project team has decided on the flow file and options file to use, the team can keep these files in a central repository pointed to by the XFLOWPATH environment variable. The variable ensures that the team members will run the same flow with the predetermined options which avoids inconsistencies.
- Installed area specified with the XILINX environment variable (Last)

If the design is an FPGA and you are using the -implement, -tsim, or -config flow types, the default flow file is fpga.flw. If the design is a CPLD and you are using the -fit or -tsim options, the default file is cpld.flw. If you are running a functional simulation for either an FPGA or CPLD using the -fsim flow type, the default flow file is fsim.flw.

XFLOW executes the programs in the order specified in the flow file. It checks the options file to find the corresponding options for each program in the flow file.

Halting XFLOW

You can manually interrupt the flow while XFLOW is in session using Control-C on your keyboard. When you halt XFLOW while PAR is in progress, you can choose one of several options. See the "Halting PAR" section of the "PAR—Place and Route" chapter for a detailed description.

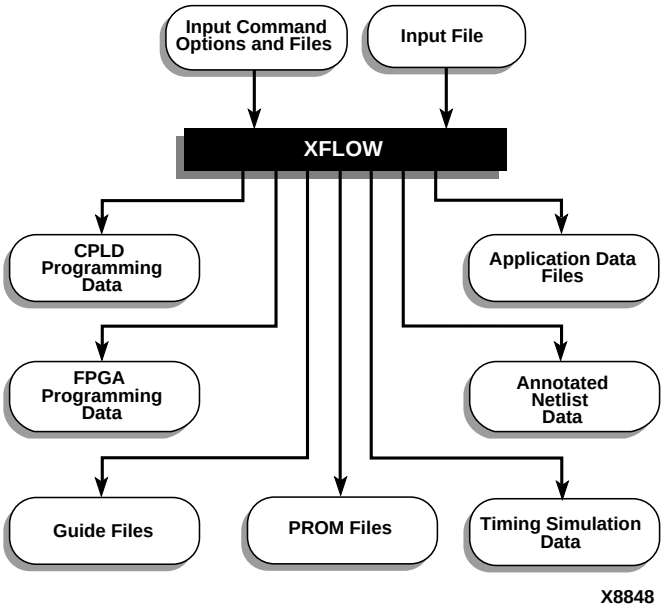


Figure 22-1 XFLOW

XFLOW Syntax

Following is the syntax for XFLOW.

```
xflow [flow type] [option file] [xflow option]
design_name
```

The combined syntax of the flow type and option file runs the Xilinx command line tools. The following table illustrates the relationship between each flow type and its option file. Xilinx provides the option files listed in the table for each flow type. You can also create your own option files.

Note This chapter uses the UNIX command line syntax. UNIX platforms use a slash (/) to specify a directory path while PCs use a backslash (\). Make sure that you use the correct syntax for your platform.

flow types	option file
-implement	fast_runtime.opt balanced.opt high_effort.opt
-tsim	See Table 22-2.
-config	bitgen.opt
-fit	balanced.opt speed.opt density.opt
-fsim	See Table 22-2.

Flow type can be any of those listed in the "Flow Types" section. If you do not specify a flow type on the command line, XFLOW defaults to the **-implement fast_runtime** flow type on FPGA devices and the **-fit balanced** flow type on CPLD devices. To run functional simulation, you must specify a flow type. See Table 22-2 for more information. Each of these flow types requires a file argument, for example, **-implement balanced.**

Note The -fsim flow type cannot be used with the -implement, -tsim, -fit, or -config flow types.

An *option file* must be specified for each flow type. See the "Option Files" section for details.

An *xflow option* can be any of the options listed in the following table:

XFLOW Options	Arguments	Description
–ed	*export_directory*	Copies export files to the *export_directory*
-f	*filename*	Reads command line argument from file
–g	*variable:value*	Specifies a global variable
–h		Displays help / usage message
–log		Specifies a log file name
–norun		Creates flow, option, and script files in the working directory and then stops.
–o	*output_filename*	Changes output file name
–p	*partname*	Specifies a part name
–rd	*report_directory*	Copies report files to the *report_directory*
–wd	*working_directory*	Specifies a working directory

Options can be listed in any particular order. Separate multiple options with spaces. The *design_name* is the only required input file. See the “Option Files” section for a description of input design file formats.

Running XFLOW

You can run XFLOW iteratively on your design. Before each run, you can edit any input file. Input files are netlists or program files. When you edit program files and rerun XFLOW, you need not change the command line syntax. XFLOW detects that an input file has changed and runs the flow from that point on. For instance, if you modify the EDIF file, XFLOW runs all the programs for that flow. If, instead, you modify the MFP file that the Floor Planner program produces, XFLOW detects that change and starts the next flow from MAP.

The examples below illustrate how XFLOW works on the first run and on successive runs.

Example 1

The following example shows how to run XFLOW for the first time using the -implement flow type.

```
xflow -implement balanced calc
```

You must have an argument for the flow type. In this example, the argument for the -implement flow type is the balanced.opt file. Notice, the filename extensions for both the balanced.opt file and the calc.edf files are not specified on the command line. XFLOW looks for the EDIF file with the design name calc, determines it is new or edited, and executes all the programs specified in the flow file.

XFLOW searches for the fpga.flw and balanced.opt files in the working directory. If these files cannot be found in the working directory, XFLOW copies the files to the working directory from the hierarchical search.

Example 2

The following example illustrates how to use the same command line syntax to rerun part of the flow. The user edits the MFP file produced by the Floorplanner and wants to rerun the flow from MAP.

```
xflow -implement balanced calc
```

The command line syntax remains the same as in Example 1. However, XFLOW detects the change in the calc.mfp file and therefore starts executing the programs in the flow file from MAP.

Note In order for XFLOW to detect input file changes on its own, you must not enter the design filename extension on the command line. If you do specify the design filename extension, it is an explicit directive to XFLOW to look at that particular input file. In the above example, you can specify the NGD extension to the design. If you enter the filename calc.ngd on the command line it explicitly directs XFLOW to use the NGD file as its input and start the flow from MAP.

Example 3

On the third run of XFLOW, the user edits the option file by modifying the Trace command line.

```
xflow -implement balanced calc
```

The command line syntax remains the same as in the earlier runs of XFLOW shown in Examples 1 and 2. In this case, XFLOW notes the change in the option file and starts the flow from the Trace program.

More Examples

The examples in this section show how to use combinations of flow types and options on different designs.

The following example shows how to use a combination of flow types to implement, configure, and perform an EDIF timing simulation on an FPGA.

```
xflow -p xcv100bg256-5 -implement balanced -tsim
generic_edif -config bitgen testclk
```

The following example shows how to use a combination of flow types to fit and perform a VHDL timing simulation on a CPLD.

```
xflow -p xc95144pq160-7 -fit balanced -tsim
generic_vhdl main_pcb
```

Flow Types

Following is a description of the flow types and how they affect the behavior of XFLOW. For a desired flow, select a combination of the -implement, -tsim, -fit, -fsim, and -config flow types.

Note The -fsim flow type must be used by itself and cannot be combined with the -implement, -tsim, -fit, or -config flow types.

You do not need to specify the complete path for option files on the command line. XFLOW searches for option files in the following hierarchy:

- Current working directory (first)
- Directories specified using XFLOWPATH
- $XILINX (last)

If you want to create your own option files, Xilinx recommends that you make a copy of an existing file, rename it option_file.opt, and then modify it.

–config (Create a BIT File for FPGAs)

-config *option_file*[**.opt**]

This flow type creates a bitstream for FPGA device configuration. The -config flow type automatically invokes the fpga.flw flow file in the $XILINX/xilinx/data directory. The flow file runs only BITGEN.

Xilinx provides the bitgen.opt option file as the only *option_file* in the $XILINX/xilinx/data directory.

Note The -config flow type requires a file name argument. There is no implied default.

Example:

The following example shows how to use a combination of flow types to implement and configure an FPGA.

```
xflow -p xcv100bg256-5 -implement balanced[.opt]
-config bitgen[.opt] testclk.edf
```

–fit (Fit a CPLD Device)

-fit *option_file*[**.opt**]

This flow type incorporates logic from a design into physical macrocell locations in a CPLD. Routing is performed automatically.

Xilinx provides three files as *option_files* in the $XILINX/epld/data directory. These files are shown in the following table.

Table 22-1 Option Files for -fit

Option Files	Description
balanced.opt	balanced between speed and density
speed.opt	optimized for speed
density.opt	optimized for density

The -fit flow type automatically copies the cpld.flw flow file from the $XILINX/epld/data directory. The flow file runs the ngdbuild, hitop, taengine, hprep sequence.

Note The -fit flow type requires a file name argument. There is no implied default.

Example:

The following example shows how to use a combination of flow types to fit and perform a VHDL timing simulation on a CPLD.

```
xflow -p xc95144pq160-7 -fit balanced[.opt] -tsim
generic_vhdl[.opt] main_pcb.edn
```

–fsim (Perform a Functional Simulation)

-fsim *option_file*[**.opt**]

This flow type performs a functional simulation for FPGA or CPLD designs.

Xilinx provides FPGA *option_files* in the $XILINX/xilinx/data directory and CPLD *option_files* in $XILINX/epld/data. The following table summarizes these option files.

Table 22-2 Option Simulation Files

Flow Name	Family	Option File	Description
VHDL Functional/ Timing Simulation	FPGA/ CPLD	generic_vhdl.opt	Generic VHDL
		active_vhdl.opt	Active VHDL
		modelsim_vhdl.opt	Modelsim VHDL
		vss_vhdl.opt	VSS VHDL
		speedwave_vhdl.opt	Speedwave VHDL
Verilog Functional/ Timing Simulation	FPGA/ CPLD	generic_verilog.opt	Generic Verilog
		modelsim_verilog.opt	Modelsim Verilog
		concept_nc_verilog.opt	Concept-NC Verilog
		concept_verilog_xl.opt	Concept Verilog-XL
		nc_verilog.opt	NC Verilog
		verilog_xl.opt	Verilog-XL
		vcs_verilog.opt	VCS Verilog
EDIF Functional/ Timing Simulation Flow	FPGA/ CPLD	generic_edif.opt	Generic EDIF
		fndtn_edif.opt	Foundation EDIF
		viewsim_edif.opt	Viewsim EDIF
		quicksim_edif.opt	Quicksim EDIF

Note The -fsim flow type requires a file name argument. There is no implied default. Also, you cannot use this flow type in conjunction with the -implement, -tsim, -config, or -fit flow types.

The following example show how to perform an EDIF functional simulation on an FPGA.

```
xflow -p xcv100bg256-5 -fsim generic_edif[.opt]
testclk.edf
```

–implement (Run FPGA implementation)

-implement *option_file*[**.opt**]

Xilinx provides three option_files in the $XILINX/xilinx/data directory. The -implement flow type automatically invokes the fpga.flw flow file in the $XILINX/xilinx/data directory. The flow file runs ngdbuild, map, trce, par, and trce.

Note The -implement flow type requires a file name argument. There is no implied default.

The following table shows the option files provided by Xilinx.

Table 22-3 Option Files for -implement

Option Files	Family	Description
fast_runtime.opt	FPGA	Runs the software tools non-timing driven. This option file provides the fastest runtimes at the expense of design performance. It is recommended for medium to slow speed designs.
balanced.opt	FPGA	Runs at PAR Effort Level 2. Operates at a level between fast_runtime.opt and high_effort.opt
high_effort.opt	FPGA	Runs the software tools timing driven at PAR Effort Level 4. High effort creates longer runtimes. It is recommended for creating designs that operate at high speeds.

The following example show how to use the -implement flow type.

```
xflow -p xcv100bg256-5 -implement balanced[.opt]
testclk.edf
```

XFLOW searches for the fpga.flw and balanced.opt files in the working directory. If these files cannot be found in the working directory, XFLOW copies the files to the working directory from either the path specified by the XFLOWPATH environment variable the install area and then executes the programs specified in the flow file.

–tsim (Perform a Timing Simulation)

-tsim *option_file*[**.opt**]

This flow type performs a timing simulation for FPGA or CPLD designs.

Xilinx provides FPGA *option_files* in the $XILINX/xilinx/data directory and CPLD *option_files* in $XILINX/epld/data. See Table 22-2 for a list of the files.

Note The -tsim flow type requires a file name argument. There is no implied default.

Example:

The following example shows how to use a combination of flow types to fit and perform a VHDL timing simulation on a CPLD.

```
xflow -p xc95144pq160-7 -fit balanced.opt -tsim
generic_vhdl.opt main_pcb.edn
```

Option Files

The options for all programs that are part of a flow are contained in option files. These files have an .opt extension. Xilinx provides option files for each flow type. Refer to the previous flow file subsections for a listing of the installed option files.

- –config (Create a BIT File for FPGAs)
- –fit (Fit a CPLD Device)
- –fsim (Perform a Functional Simulation)
- –implement (Run FPGA implementation)

- –tsim (Perform a Timing Simulation)

The option files are located in the $XILINX/xilinx/data or $XILINX/epld/data directories.The option file data is in ASCII format, which can be edited.

Option File Structure and Content

For each program in the flow file, the option file has a corresponding program block. This block lists the default command line options that are enabled. Preceding each block is a comment section with help messages to obtain the list of all command line options for a program.

Options in the option file can be switches, files, or parameter files.

- Switches

  Switches that do not require any arguments, that is, the switch enables or disables a specific operation. For example. -r

  Switches that require one or more arguments that are strings or integers. For example, -pl 5 or -cm area.

  Switches that accept values in the format *option value*:*value*. For example, -g DonePin:PULLUP;

- Files

  This category includes command line options that are just file names.

  Example: calc.pcf;

- Parameter files

  This category specifies parameters for programs in a flow. All the parameters are written into a file.

  Example:

  CTL file for program hitop for CPLD devices. The option file for CPLD flow has the following entry for program "hitop" to create the CTL file.

```
Program hitop
     -f <design>.ngd;
     -d <design>;
```

```
        -l <design>.log
        ParamFile: <design>.ctl
            "DT_SYNTHESIS:TRUE";
            "MC9500_INPUT_LIMIT: 36";
            "GSR_OPT; TRUE";
        End ParamFile
    End Program
    hitop
```

For this example, XFLOW creates a CTL file, design.ctl with all the parameters listed in the option file.

Option File Sample

Following is an option file, balanced.opt, that is used with the -implement flow type. For the most up-to-date version of the file, see the file located in $XILINX/xilinx/data.

```
FLOWTYPE = FPGA;
####################################################################
## Filename: balanced.opt
##
## Option File For Xilinx FPGA Implementation Flow
## Version: 3.1.1
####################################################################


# Options for Translator
#
# Type "ngdbuild -h" for a detailed list of ngdbuild command line
options
#
Program ngdbuild
-p <partname>;      #  Partname  to  use  -  picked  from  xflow
commandline
-nt timestamp;      # NGO File generation. Regenerate only when
                    # source netlist is newer than existing
                    # NGO file (default)
<userdesign>;       # User design - pick from xflow command line
```

```
<design>.ngd;       # Name of NGD file. Filebase same as design
filebase
End Program ngdbuild


#
# Options for Mapper
#
# Type "map -h <arch>" for a detailed list of map command line
options
#
Program map
-o <design>_map.ncd;# Output Mapped ncd file
<inputdir><design>.ngd;# Input NGD file
<inputdir><design>.pcf;# Physical constraints file
END Program map

#
# Options for Post Map Trace
#
# Type "trce -h" for a detailed list of trce command line options

#
Program post_map_trce
-e 3;               # Produce error report limited to 3 items per
constraint
-o <design>_map.twr;# Output trace report file
<inputdir><design>_map.ncd; # Input mapped ncd
<inputdir><design>.pcf;     # Physical constraints file
END Program post_map_trce
#
# Options for Place and Route
#
# Type "par -h" for a detailed list of par command line options

#
Program par
<design>_map.ncd;   # Input mapped NCD file
<inputdir><design>.ncd; # Output placed and routed NCD
<inputdir><design>.pcf; # Input physical constraints file
END Program par
```

```
#
# Options for Post Par Trace
#
# Type "trce -h" for a detailed list of trce command line options

#
Program post_par_trce
-e 3;               # Produce error report limited to 3 items per
constraint
-o <design>.twr;    # Output trace report file
<inputdir><design>.ncd; # Input placed and routed ncd
<inputdir><design>.pcf; # Physical constraints file
END Program post_par_trce
```

XFLOW Options

This section describes the xflow options. These options can be used at the command line with any of the flow types and their arguments.

–ed (Copy Files to Export Directory)

–ed *export_directory*

This option copies files specified in the export section of a flow file to the *export_directory*. The default is the working directory. See the "Flow Files" section for a description of the export section of the flow file.

If you use the -ed *export_directory* option with the -wd *working_directory* option and do not specify an absolute pathname for the *export_directory*, the directory is placed underneath the *working_directory*. For example,

```
xflow -p xcv100bg256-5 -implement balanced.opt -wd
sub3 -ed /usr/export3 testclk.edf
```

The export3 directory is created if it does not already exist underneath the sub3 directory.

If you do not want the export directory to be a subdirectory of the working directory, enter an absolute pathname. For example,

```
xflow -p xcv100bg256-5 -implement balanced.opt -wd
sub3 -ed /usr/export3 testclk.edf
```

–g (Specify a Global Variable)

-g *variable:value*

You can specify a global variable at the command line or in the flow file. If global variables are specified in both places, the command line takes precedence over the flow file. For an example of global variables in the flow file, see the sample flow file in the "Flow Files" section.

The following example shows how to specify a global variable at the command line.

xflow -implement balanced -g*$simulation_output : time_sim* **calc**

–h (Help)

The -h option displays a list of valid options with brief descriptions.

–log (Specify Log File)

The -log option allows you to specify a log filename at the command line. XFLOW writes the log file to the working directory after each run. By default, the log filename is xflow.log.

–norun (Creates a Script File)

By default, XFLOW begins executing programs that are enabled in the flow file. If you do not want program execution to begin, specify the -norun option. XFLOW then creates test flow and option files listing the command lines for all the enabled program and then stops.

The following example shows how to use the -norun option to create the initial flow and options files in the working directory and then stop.

```
xflow -p xcv100bg256-5 -implement balanced.[opt]
-norun testclk.edf
```

This example copies the balanced.opt and fpga.flw files to the current directory. The example command also creates the following script file:

```
##########################################
# Script file to run the flow
#
##########################################
#
# Command line for ngdbuild
#
ngdbuild -p xcv100bg256-5 -nt timestamp /home/
xflow_test/testclk.edf testclk.ngd
#
# Command line for map
#
map -o testclk_map.ncd testclk.ngd testclk.pcf
#
# Command line for par
#
par -w -ol 2 -d 0 testclk_map.ncd testclk.ncd
testclk.pcf
#
# Command line for post_par_trce
#
trce -e 3 -o testclk.twr testclk.ncd testclk.pcf
```

–o (Change Output File Name)

-o *output_filename*

This option allows you to change the output file base name. If you do not specify this option, the output file name will have the base name of the input file.

The following example show how to use the -o option to change the base name of output files.

```
xflow -p xcv100bg256-5 -implement balanced[.opt] -o
newname testclk.edf
```

All output files have the base name “newname” instead of “testclk”.

–p (Enter a Part Name)

-p *partname*

The –p option allows you to specify a device. For a list of valid options, see the “Part Numbers in Commands” section of the “Introduction” chapter.

For FPGA part types, you must designate a part name with a package name. If you do not, XFLOW halts at map indicating that a package needs to be specified. (You can obtain package names for installed devices using the PARTGEN command with the -i option.)

If the -p option is not specified, one of the following occurs:

- XFLOW searches for the part name in the input design file. If XFLOW finds a part number, it uses that number as the target device for the design.
- If XFLOW does not find a part number in the design input file, it prints an error message indicating that a part number is missing.

Note If the design is a CPLD, either the part number or the family name can be specified.

The following example show how to use the -p option for a Virtex design.

```
xflow -p xcv100bg256-5 -implement high_effort.opt
testclk.edf
```

–rd (Copy Report Files)

–rd *report_directory*

The -rd option copies the reports specified in the report section of the of the flow file to the *report_directory*. The default *report_directory* is the working directory.

If you use the -rd *report_directory* option with the -wd *working_directory* option and do not specify an absolute pathname for the *report_directory*, the directory is placed underneath the *working_directory*. For example,

```
xflow -p xcv100bg256-5 -implement balanced.opt -wd
sub3 -rd report3 testclk.edf
```

The report3 directory is created, if it does not already exist, underneath the sub3 directory.

If you do want the report directory to be a subdirectory of the working directory, enter an absolute pathname. For example,

```
xflow -p xcv100bg256-5 -implement balanced.opt -wd
sub3 -rd /usr/report3 testclk.edf
```

Refer to the following for a list of FPGA and CPLD report files:

- Table 22-6
- Table 22-7

–wd (Specify a Working Directory)

–wd *working_directory*

The -wd option defines a working directory. The default is the current directory. If no path is specified for the working_directory, then the directory is created, if it does not already exist, in the current working directory and all generated files are placed in the new directory. For example, assume that your current working directory is named “project”, your design name is “test_clock.edf” and you enter the following command:

```
xflow -p xcv100bg256-5 -fsim generic_edif.opt -wd sub1
testclk.edf
```

The directory sub1 is created, if it does not already exist, and all the files generated from XFLOW are placed in the sub1 directory. Following is a list of the files for the example.

- fsim.flw
- netlist.lst
- test_clock.ngo
- func_sim.edn
- test_clock.bld
- xflow.log
- generic_edif.opt
- test_clock.ngd
- xflow.scr

You can also enter an absolute path for a working_directory. Following is an example for an existing directory, /usr/project1.

```
xflow -p xcv100bg256-5 -fsim generic_edif.opt -wd /
usr/project1 testclk.edf
```

In this example, all generated files are placed in /usr/project1.

Input Files

The inputs to XFLOW are a user input file, a flow file, and one or more option files.The flow file is automatically utilized when specifying a flow type—implement, tsim, config, fit, or fsim.

User Input Design File

The only required input file to XFLOW is an input design file. This design file can be an EDIF or XNF netlist, a PLD file, an NGD file, or NCD file. The following table indicates valid input files. The input file must be in one of these formats.

An input netlist file usually has a top-level module and several sub-design modules. The top-level file must be in one of the supported formats. The sub-design modules can be in other formats such as Verilog and VHDL.

Table 22-4 Valid Input Files

File Type	Syntax	Location
EDIF Netlist	*.edf *.edn *.edif *.sedif	Current working directory
XNF Netlist	*.xnf *.xtf *.sxnf	Current working directory
PLD File	*.pld	Current working directory
NGD File	*.ngd	Current working directory
NCD File	*.ncd	Current working directory

The UCF file allows you to specify constraints independently of an input netlist file. There can only be one UCF file per design. The UCF file will be read automatically by NGDBuild if the file resides in the same directory and has the same base name as the input design file. If the UCF file must have a different base name than an input netlist file, you can modify the -uc option for NGDBuild in an option file to read the UCF file.

All other input files are specified as arguments to the options. These option files are described in the “Option Files” section.

NGD files or NCD files can also be used as input files if you want to start the flow at an intermediate point such as MAP or PAR, rather than at the beginning stage with an EDIF or XNF netlist.

Flow Files

The following subsections describe the flow file and provide an example.

Description

All designs targeted for Xilinx devices follow a flow. A flow is a sequence of programs that are automatically invoked to implement, configure, and simulate a design. For example, to prepare an FPGA design for VHDL timing simulation, run the design through ngdbuild, map, par, ngdanno, and ngd2vhdl program flow. This sample flow requires the use of the -implement and -tsim flow types.

If you select either the -implement, -fit, -config, -tsim, or -fsim flow type, XFLOW automatically invokes one of the XIlinx flow files. The following table shows which flow files are invoked for each flow type.

Table 22-5 Xilinx Flow Files

Family	Flow Name	Flow Type	Flow Phase	Programs
FPGA	fpga.flw	-implement	Implementation	ngdbuild, map, trce, par, trce
		-tsim	Timing Simulation	ngdanno, ngd2edif, ngd2ver, ngd2vhdl
		-config	Configuration	bitgen
CPLD	cpld.flw	-fit	Fit	ngdbuild, hitop, taengine, hprep6
		-tsim	Timing Simulation	tsim, ngd2edif, ngd2ver, ngd2vhdl
FPGA / CPLD	fsim.flw	-fsim	Functional Simulation	ngdbuild, ngd2edif, ngd2ver, ngd2vhdl

Note Invoked programs depend on which option file is used with each flow type. See the “Option Files” section for details about option files.

The flow file, which is ASCII format, contains the following information.

- ExportDir

  The directory in which to copy the outputs of individual programs in the flow. The output files are specified under the Exports option in the program block. ExportDir defaults to your working directory. You can also specify the export directory using the -ed command line option. The command line overrides any Export directory in a flow file.

- ReportDir

  The directory in which to copy the report files generated by the programs in the flow. The report files are specified under the Reports option in the program block. ReportDir defaults to your working directory. You can also specify the report directory using the -rd command line option. The command line overrides any Report directory in a flow file.

- Program block for each executable in the flow

  Each program has the following information

  - **Program** *program_name*

    The name of the program, for example, ngdbuild. The Program statement is the first line of the Program block. The last line of a Program block is End Program.

  - **Flag: ENABLED | DISABLED**

    ENABLED: Only run the program if there are options in the options file.

    DISABLED: Do not run the program even if there are options in the options file.

  - **Input:***filename*

    The name of the input file for the program. For example, for the map program, the input file is a design.ngd file.

  - **Executable:***executable_name*

    A program block can be defined with or without the Executable construct. The Executable construct is useful when you want to define multiple option blocks for the same program.

Case 1:

If a program is defined with the Executable construct, XFLOW uses the *executable_name* to build the program's command line. The program's option block in the option file has the same *program_name*. The *program_name* masks the executable name so that you can define multiple option blocks for the same program allowing you to control which programs are executed

For example, if you want to run TRCE, once after MAP and then again after PAR. The program name can be given any name you want. Within the block, the Executable construct will specify the correct name of the program. When XFLOW builds the command line for the program, it uses the name specified within the Executable construct. When you define the options for TRCE in the option file, there will be two entries and XFLOW will choose the correct one.

Example:

Program post_map_trce
Flag: ENABLED
Executable: trce
Input: *design_map*.ncd
Exports: *design*.twr
End Program post_map_trce

Case 2:

If a program block is defined without the Executable construct, XFLOW uses the *program_name* to build the command line

- **Exports:** *exported_files_list*

  List of program output files to copy into ExportDir

- **Reports:** *list_of_report_files*

  List of program generated report files to copy into ReportDir

- Variable Assignments **$***variable_name=value***;**

  Variable assignments can be used in the option files. During runtime, variables are substituted with their assigned values. You can use variables to specify the base name for the reports and program input or output files. The base name for functional simulation is func_sim. The base name for timing simulation is time_sim. Defined variables have global scope.

- **End Program** *program_name*

  The last line of a Program block is End Program.

- User command block

  You can use this block to run programs or scripts between Xilinx programs. For example, if you want to run a script after map, you can add a User Command Block in the fpga.flw file as follows:

```
UserCommand
   Cmdline: “myscript.csh”
   End UserCommand
```

  Following are some more examples of User Command Blocks:

```
UserCommand
   Cmdline:"printenv";
   End UserCommand
UserCommand
   Cmdline:"ls -ltr";
   End UserCommand
UserCommand
Cmdline:"ls -l calc.ngo";#This works if calc.ngo
                        #exists.
                        # If calc.ngo does not
   exist,
                        # the ls command will
                        # return a non zero
                        # to xflow and xflow will
```

```
                                    # end and give an
                                    # error message.
End UserCommand
UserCommand
Cmdline:"/home/test/xflow/userscript.csh";
End UserCommand
```

Following are some examples of User Command Blocks that do not work.

```
UserCommand
   Cmdline:"ls -l *.ngo";   # doesn't handle the
   "*"
   End UserCommand
UserCommand
   Cmdline:"/bin/rm -rf $MYVAR/calc.ngo";
   # doesn't handle the "$"
   End UserCommand
UserCommand
   Cmdline:"if ( -e *.ngo ) /bin/rm -rf *.ngo";
   # doesn't hamdle the "("
   End UserCommand
```

Flow File Example

Following is an example fpga.flw file. For the most up-to-date version of the file, see the file located in $XILINX/xilinx/data.

```
#######################################################
## FileName: fpga.flw
##
## Flow File to run Xilinx FPGA Flow
## Version: 3.1.1
#######################################################
ExportDir: <workdir>; # Directory to copy program outputs
ReportDir: <workdir>; # Directory to copy program reports
```

```
#
#Global user-defined variables
#
Variables
$simulation_output = time_sim;
End variables

#
#Flow Info for XST
#
#Program xst
#Flag: ENABLED
#Input: <synthdesign>
#Triggers: <design>.cst;
#Exports: <design>_xst.edn;
#End Program xst

#
# Flow Info for Translator
#
Program ngdbuild
Flag : ENABLED;
Input: <userdesign>
Triggers: <design>.ucf, <design>.urf, <design>.ncf, netlist.lst;
Exports : <design>.ngd;
Reports : <design>.bld;
End Program ngdbuild

#
# Flow Info for Mapper
#
Program map
Flag: ENABLED;
Input: <design>.ngd;
Triggers: <design>.mfp;
Exports : <design>_map.ncd;
Reports : <design>_map.mrp;
End Program map

#
# Flow Info for Post Map Trace
```

```
#
Program post_map_trce
Flag: DISABLED;
Executable: trce;
Input: <design>_map.ncd;
Reports: <design>.twr; <design>_map.tsi;
End Program post_map_trce

#
# Flow Info for Place and Route
#
Program par
Flag : ENABLED;
Input: <design>_map.ncd;
Triggers: <design>.pcf;
Exports : <design>.ncd;
Reports: <design>.par <design>.dly, <design>.pad;
End Program par

#
# Flow Info for Post Par Trace
#
Program post_par_trce
Flag: ENABLED;
Executable: trce;
Input: <design>.ncd;
Reports: <design>.twr; <design>.tsi;
End Program post_par_trce

#
# Flow Info for Annotator
#
Program ngdanno
Flag: ENABLED;
Input: <design>.ncd;
Exports: <design>.nga;
End Program ngdanno

#
# Flow Info for EDIF Netlist Writer
#
Program ngd2edif
```

```
Flag: ENABLED;
Input: <design>.nga;
$input_extension = nga;
Exports: $simulation_output.edn $simulation_output.xmm;
End Program ngd2edif

#
#Flow Info for Mentor Quicksim EDIF
#
Program ngd2edif_mentor
Flag: ENABLED
Executable: ngd2edif;
Input: <design>.nga;
$input_extension = nga;
Exports: <design>.edn, <design>.xmm;
End Program ngd2edif_mentor

#
# Flow Info for Verilog Netlist Writer
#
Program ngd2ver
Flag: ENABLED;
Input: <design>.nga;
$input_extension = nga;
Exports:  $simulation_output.v $simulation_output.sdf,
          $simulation_output.pin, $simulation_output.tv;
End Program ngd2ver

#
# Flow Info for VHDL Netlist Writer
#
Program ngd2vhdl
Flag: ENABLED;
Input: <design>.nga;
$input_extension = nga;
Exports:  $simulation_output.vhd $simulation_output.sdf,
          $simulation_output.pin, $simulation_output.tvhd;
End Program ngd2vhdl

#
# Flow Info for Bitgen
#
```

```
Program bitgen
Flag: ENABLED;
Input: <design>.ncd;
Exports: <design>.bit, <design>.ll, <design>.msk, <design>.rbt;
Reports: <design>.bgn, <design>.drc;
End Program bitgen
#
```

Output Files

Output files from XFLOW may be generated for the following:

- FPGA programming data
- CPLD programming data
- Annotated netlist data—These files are only generated if you run the timing simulation flows
- Timing simulation data—These files are only generated if you run the timing simulation flows. They are the primary output of the back-annotation process.
- Functional simulation data—These files are only generated if you run the functional simulation flows.
- Guide data
- Reports

The following tables illustrate specific files that are generated for each type.

Table 22-6 FPGA Programming Data

File Type	Syntax	Location
Bitstream	*design*.bit	Current working directory Export directory
LL file	*design*.ll	Current working directory Export directory
Rawbits file	*design*.rbt	Current working directory Export directory

The *design*.ll file is generated if the -l option is set for bitgen in the option file.

The *design*.rbt file is created if the -b option is set for bitgen in the option file.

Table 22-7 CPLD Programming Data

File Type	Syntax	Location
JEDEC file	*design*.jed	Current working directory Export directory
Design file	*design*.vm6	Current working directory Export directory

Note that VM6 files from previous releases cannot be used as inputs to 3.1i downstream processes. You must rerun the fitter using the 3.1i software to create a new VM6 file before performing any of the following tasks:

- Creating a JEDEC programming file (.jed file from the hprep program
- Creating a timing report (.tim file from the taengine)
- Running the Timing Analyzer
- Creating a timing simulation model (.nga file from the tsim program) for input to NGD2EDIF, NGD2VHDL, or NGD2VER.

Table 22-8 Annotated Netlist Data

File Type	Syntax	Location
EDIF netlist	time_sim.edn	Current working directory Export directory
XNF netlist	time_sim.xnf	Current working directory Export directory
XMM file	time_sim.xmm	Current working directory Export directory

Annotated netlist files are only generated if you run timing simulation flows.

Table 22-9 Functional Simulation Data

File Type	Syntax	Location
EDIF simulation file	func_sim.edif	Current working directory Export directory
Verilog simulation file	func_sim.v	Current working directory Export directory
Verilog test vector file	func_sim.tv	Current working directory Export directory
VHDL simulation file	func_sim.vhd	Current working directory Export directory
VHDL test vector file	func_sim.tvhd	Current working directory Export directory

Table 22-10 Timing Simulation Data

File Type	Syntax	Location
EDIF simulation file	time_sim.edif	Current working directory Export directory
Verilog simulation file	time_sim.v	Current working directory Export directory
Verilog test vector file	time_sim.tv	Current working directory Export directory
VHDL simulation file	time_sim.vhd	Current working directory Export directory
VHDL test vector file	time_sim.tvhd	Current working directory Export directory
SDF simulation file	time_sim.sdf	Current working directory Export directory

Timing simulation data files are only generated if you run timing simulation flows. These files are the primary output of back-annotation.

Table 22-11 Guide Data

File Type	Syntax	Location
FPGA guide file	*.ncd	Current working directory Export directory
CPLD guide file	*.gyd	Current working directory Export directory

Table 22-12 Application Data Files

File Type	Syntax	Location
XFLOW log file	xflow.log	Current working directory Export directory
XFLOW script file	xflow.scr	Current working directory Export directory
XFLOW history file	xflow.his	Current working directory Export directory

XFLOW generates various reports depending on which command line options are used. The following tables indicate the types of generated reports.

Table 22-13 FPGA Report Files

Report	Flow Type	Location
Translation Report (design.bld)	-implement	Current Working Directory Report Director
Mapping (design_map.mrp)		Current Working Directory Report Director
Logic Level Timing (design_map.twr)		Current Working Directory Report Director
Place and Route (design.par)		Current Working Directory Report Director
Pad (design.pad)		Current Working Directory Report Director

Table 22-13 FPGA Report Files

Report	Flow Type	Location
Asynchronous Delay (design.dly)		Current Working Directory Report Director
Post Layout Timing (design.twr)		Current Working Directory Report Director
BitGen (design.bgn)	-config	Current Working Directory Report Director

Table 22-14 CPLD Report Files

Report	Flow Type	Location
Translation (design.bld)	-fit	Current Working Directory Report Directory
Fitting (design.rpt)		Current Working Directory Report Directory
Post Layout Timing (design.tim)		Current Working Directory Report Directory

Appendix A

Xilinx Development System Files

This appendix gives an alphabetic listing of the files used by the Xilinx Development System.

Name	Type	Produced By	Description
ALF	ASCII	NGDAnno	Log file containing information about an NGDAnno run
ARF	ASCII	NGDAnno	Report file containing information about lost instance or net names
BIT	Data	BitGen	Download bitstream file for devices containing all of the configuration information from the NCD file
BGN	ASCII	BitGen	Report file containing information about a BitGen run
BLD	ASCII	NGDBuild	Report file containing information about an NGDBuild run, including the subprocesses run by NGDBuild
DATA	C File	TRCE	File created with the -stamp option to TRCE that contains timing model information
DC	ASCII	Synopsys FPGA Compiler	Synopsys setup file containing constraints read into the Xilinx Development System
DLY	ASCII	PAR	File containing delay information for each net in a design
DRC	ASCII	BitGen	Design Rule Check file produced by BitGen

Name	Type	Produced By	Description
EDIF (various file extensions)	ASCII	CAE vendor's EDIF 2 0 0 netlist writer.	EDIF netlist. The Xilinx Development System accepts an EDIF 2 0 0 Level 0 netlist file
EDN	ASCII	NGD2EDIF	Default extension for an EDIF 2 0 0 netlist file
EPL	ASCII	FPGA Editor	FPGA Editor command log file. The EPL file keeps a record of all FPGA Editor commands executed and output generated. It is used to recover an aborted FPGA Editor session.
EXO	Data	PROMGen	PROM file in Motorola's EXORMAT format
FLW	ASCII	Provided with software	File containing command sequences for XFLOW programs
fpga_editor.ini	ASCII	Xilinx software	Script that determines what FPGA Editor commands are performed when the FPGA Editor starts up
fpga_editor_user.ini	ASCII	Xilinx software	Supplement to the fpga_editor.ini file used for modifying or adding to the fpga_editor.ini file
GYD	ASCII	CPLD fitter	CPLD guide file
HEX	Hex	PROMGen Command	Output file from PROMGEN that contains a hexadecimal representation of a bitstream
ITR	ASCII	PAR	Intermediate failing timespec summary from routing
JED	JEDEC	CPLD fitter	Programming file to be downloaded to a device
LOG	ASCII	NGD2VER NGD2VHDL	Log file containing all the messages generated during the execution of NGD2VER (ngd2ver.log) or NGD2VHDL (ngd2vhdl.log)
LCA	ASCII	Xilinx Development System	Mapped file of an earlier release Xilinx design

Name	Type	Produced By	Description
L2N	ASCII	LCA2NCD	Report file containing information about an LCA2NCD run
LL	ASCII	BitGen	Optional ASCII logic allocation file with an .ll extension. The logic allocation file indicates the bitstream position of latches, flip-flops, and IOB inputs and outputs.
MEM	ASCII	User (with text editor) LogiBLOX	User-edited memory file that defines the contents of a ROM
MCS	Data	PROMGen	PROM-formatted file in Intel's MCS-86 format
MDF	ASCII	MAP or LCA2NCD	A file describing how logic was decomposed when the design was mapped. The MDF file is used for guided mapping.
MFP	ASCII	Floorplanner	Map Floorplanner File, which is generated by the Floorplanner, specified as an input file with the –fp option. The MFP file is essentially used as a guide file for mapping.
MOD	ASCII	TRCE	File created with the –stamp option in TRCE that contains timing model information
MRP	ASCII	MAP	MAP report file containing information about a technology mapper command run
MSK	Data	BitGen	File used to compare relevant bit locations when reading back configuration data contained in an operating Xilinx device
NCD	Data	Mappers, LCA2NCD, PAR, FPGA Editor	Flat physical design database correlated to the physical side of the NGD in order to provide coupling back to the user's original design

Name	Type	Produced By	Description
NCF	ASCII	CAE Vendor toolset	Vendor-specified logical constraints files
NGA	Data	NGDAnno	Back-annotated mapped NCD file
NGC	Binary	LogiBLOX	File containing the implementation of a module in the design
NGD	Data	NGDBuild	Generic Database file. This 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	Data	MAP	File containing all of the data in the input NGD file as well as information on the physical design produced by the mapping. The NGM file is used for back-annotation.
NGO	Data	Netlist Readers	File containing a logical description of the design in terms of its original components and hierarchy
NMC	Binary	FPGA Editor	Xilinx physical macro library file containing a physical macro definition that can be instantiated into a design
OPT	Text	Input file option	Option file used by XFLOW
PAD	ASCII	PAR	File containing a listing of all I/O components used in the design and their associated primary pins
PAR	ASCII	PAR	PAR report file containing execution information about the PAR command run. The file shows the steps taken as the program converges on a placement and routing solution

Name	Type	Produced By	Description
PCF	ASCII	MAP, FPGA Editor	File containing physical constraints specified during design entry (that is, schematics) and constraints added by the user
PRM	ASCII	PROMGen	File containing a memory map of a PROM file showing the starting and ending PROM address for each BIT file loaded
RBT	ASCII	BitGen	“Rawbits" file consisting of ASCII ones and zeros representing the data in the bitstream file
RPT	ASCII	PIN2UCF	Report file generated by PIN2UCF when conflicting constraints are discovered. The name is pinlock.rpt.
RCV	ASCII	FPGA Editor	FPGA Editor recovery file
SCR	ASCII	FPGA Editor or XFLOW	FPGA Editor or XFLOW command script file
SDF	ASCII	NGD2VER, NGD2VHDL	File containing the timing data for a design. Standard Delay Format File
TDR	ASCII	DRC	Physical DRC report file
TEK	Data	PROMGen	PROM-formatted file in Tektronix’s TEKHEX format
TV	ASCII	NGD2VER	Verilog test fixture file
TVHD	ASCII	NGD2VHDL	VHDL testbench file
TWR	ASCII	TRACE	Timing report file produced by TRACE
UCF	ASCII	User (with text editor)	User-specified logical constraints files
URF	ASCII	User (with text editor)	User-specified rules file containing information about the acceptable netlist input files, netlist readers, and netlist reader options
V	ASCII	NGD2VER	Verilog netlist
VHD	ASCII	NGD2VHDL	VHDL netlist

Name	Type	Produced By	Description
VM6	Design	CPLD Fitter	Output file from fitter
XMM	ASCII	NGD2EDIF	File defining the initial contents of the RAMs in the design for a simulator
XNF	ASCII	Previous releases of Xilinx Development System, CAE vendor toolsets	Xilinx netlist format file
XTF	ASCII	Previous releases of Xilinx Development System	Xilinx netlist format file
XPI	ASCII	PAR	File containing PAR run summary

Appendix B

EDIF2NGD, XNF2NGD, and NGDBuild

This appendix describes the netlist reader programs, EDIF2NGD and XNF2NGD, and how these programs interact with NGDBuild. The appendix contains the following sections.

- "EDIF2NGD"
- "XNF2NGD"
- "NGDBuild"
- "Netlister Launcher"
- "File Names and Locations"

EDIF2NGD

This program is compatible with the following families.

- Spartan / XL / -II
- Virtex / -E / -II
- XC9500 / XL / XV
- XC4000E / L / EX / XL / XV / XLA
- XC3000A / L
- XC3100A / L
- XC5200

The EDIF2NGD program allows you to read an EDIF (Electronic Design Interchange Format) 2 0 0 file into the Xilinx Development System toolset. EDIF2NGD converts an industry-standard EDIF netlist to an NGO file—a Xilinx-specific format. The EDIF file includes the hierarchy of the input schematic. The output NGO file is

a binary database describing the design in terms of the components and hierarchy specified in the input design file. The following figure shows the flow through EDIF2NGD.

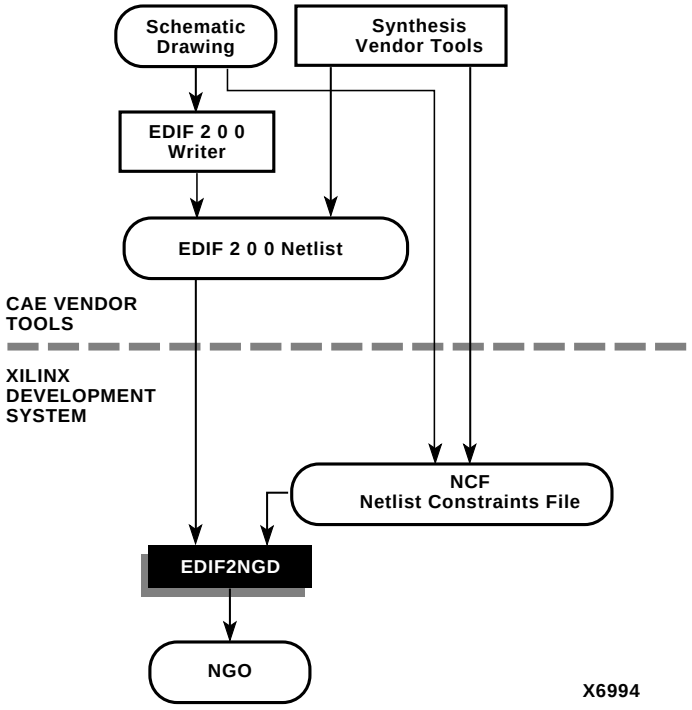


Figure B-1 EDIF2NGD Design Flow

The NGO file can be converted to an NGD file using the NGDBuild program. The NGD file can be mapped into an NCD file, which can then be placed and routed.

The input file must be a Level 0 EDIF netlist, as defined in the EDIF 2 0 0 specification. The Xilinx Development System toolset can understand EDIF files developed using components from any of these libraries.

- Xilinx Unified Libraries (described in the *Libraries Guide*)
- XSI (Xilinx Synopsys Interface) Libraries
- Any Xilinx physical macros you create

Note Xilinx tools do not recognize Xilinx Unified Libraries components defined as macros; they only recognize the primitives from this library. The third-party EDIF writer must include definitions for all macros.

You can run EDIF2NGD in the following ways.

- From the Design Manager / Flow Engine with the Translate step
- Automatically from NGDBuild
- From the UNIX or DOS command line, as described in the following sections

Note When creating nets or symbols names, do not use reserved names. Reserved names are the names of symbols for primitives and macros in the *Libraries Guide* and net names GSR, RESET, GR, and PRELOAD. If you used these names, EDIF2NGD issues an error.

EDIF2NGD Syntax

The following command reads your EDIF netlist and converts it to an NGO file.

edif2ngd [*options*] *edif_file ngo_file*

options can be any number of the EDIF2NGD options listed in the "EDIF2NGD Options" section. They do not need to be listed in any particular order. Separate multiple options with spaces.

edif_file is the EDIF 2 0 0 input file to be converted. The file must have an extension. If the file has an extension other than .edn, you must enter the extension as part of *edif_file*. If you enter a file name with no extension, EDIF2NGD looks for a file with an .edn extension and the name you specified.

Note For EDIF2NGD to read a Mentor Graphics EDIF file, you must have installed the Mentor Graphics software component on your system. Similarly, to read a Cadence EDIF file, you must have installed the Cadence software component.

ngo_file is the output file in NGO format. The output file name, its extension, and its location are determined in the following way.

- If you do not specify an output file name, the output file has the same name as the input file, with an .ngo extension.
- If you specify an output file name with no extension, EDIF2NGD appends the .ngo extension to the file name.
- If you specify a file name with an extension other than .ngo, you get an error message and EDIF2NGD does not run.
- If you do not specify a full path name, the output file is placed in the directory from which you ran EDIF2NGD.

If the output file exists, it is overwritten with the new file.

EDIF2NGD Files

This section describes the EDIF2NGD input and output files.

Input Files

EDIF2NGD uses the following files as inputs.

- EDIF file—This is an EDIF 2 0 0 netlist file. The file must be a Level 0 EDIF netlist, as defined in the EDIF 2 0 0 specification.
- NCF file—This Netlist Constraints File is produced by a vendor toolset and contains constraints specified within the toolset. EDIF2NGD reads the constraints in this file and adds the constraints to the output NGO file.

  EDIF2NGD reads the constraints in the NCF file if the NCF file has the same base name as the input EDIF file and an .ncf extension. The name of the NCF file does not have to be entered on the EDIF2NGD command line.

Output Files

The output of EDIF2NGD is an NGO file—a binary file containing a logical description of the design in terms of its original components and hierarchy.

EDIF2NGD Options

This section describes the EDIF2NGD command options.

–a (Add PADs to Top-Level Port Signals)

The –a option adds PAD properties to all top level port signals. This option is necessary if the EDIF2NGD input is an EDIF file in which PAD symbols were translated into ports. If you do not specify a –a option for one of these EDIF files, the absence of PAD instances in the EDIF file causes EDIF2NGD to read the design incorrectly. Subsequently, MAP interprets the logic as unused and removes it.

In all Mentor Graphics and Cadence EDIF files PAD symbols are translated into ports. For EDIF files from either of these vendors, the –a option is set automatically; you do not have to enter the –a option on the EDIF2NGD command line.

–f (Execute Commands File)

```
-f command_file
```

The –f option executes the command line arguments in the specified *command_file*. For more information on the –f option, see the "–f Option" section of the "Introduction" chapter.

–l (Libraries to Search)

```
-l libname
```

The –l option specifies a library to search when determining what library components were used to build the design. This information is necessary for NGDBuild, which must determine the source of the design's components before it can resolve the components to Xilinx primitives.

You may specify multiple –l options on the command line. Each must be preceded with –l; you cannot combine multiple *libname* specifiers after one –l. For example, **–l xilinxun synopsys** is not acceptable, while **–l xilinxun –l synopsys** is acceptable.

The allowable entries for *libname* are the following.

- **`xilinxun`** `(For Xilinx Unified library)`
- **`synopsys`**

Note You do not have to enter xilinxun with a –l option. The Xilinx Development System tools automatically access these libraries. You do not have to enter synopsys with a –l option if the EDIF netlist contains an author construct with the string "Synopsys." In this case, EDIF2NGD automatically detects that the design is from Synopsys.

–p (Part Name)

`-p` *`part`*

The –p option specifies the part into which your design is implemented. The –p option can specify an architecture only, a complete part specification (device, package, and speed), or a partial specification (for example, device and package only).

The syntax for the –p option is described in the "Part Numbers in Commands" section of the "Introduction" chapter. Examples of *part* entries are **XCV50-TQ144** and **XCV50-TQ144-5**.

If you do not specify a part when you run EDIF2NGD, you will have to specify one when you run NGDBuild.

You can also use the –p option to override a part name in the input EDIF netlist or a part name in an NCF file.

–r (Ignore LOC Properties)

The –r option filters out all location properties (LOC=) from the design. This option can be used when you are migrating to a different device or architecture, because locations in one architecture do not match locations in another.

XNF2NGD

This program is compatible with the following families.

- Spartan / XL / -II
- XC9500 / XL / XV
- XC4000E / L / EX / XL / XV / XLA
- XC3000A / L
- XC3100A / L
- XC5200

Note XNF primitives are not defined for the Virtex families, and XNF files created for Virtex families are rejected by XNF2NGD. However, if you have XNF netlists that were created for the XC3000, XC4000E, or XC5200 architectures, you can include these XNF netlists in a design that you target to a Virtex device.

XNF2NGD allows you to read a Version 6.1 XNF (Xilinx Netlist Format) file into the Xilinx Development System toolset. XNF2NGD converts an XNF file to an NGO file, which is a binary database describing the netlist in terms of Xilinx components. After you convert the XNF file to an NGO file, you run NGDBuild to create an NGD file, which expands the design to include a description reduced to Xilinx primitives. The following figure shows the flow through XNF2NGD.

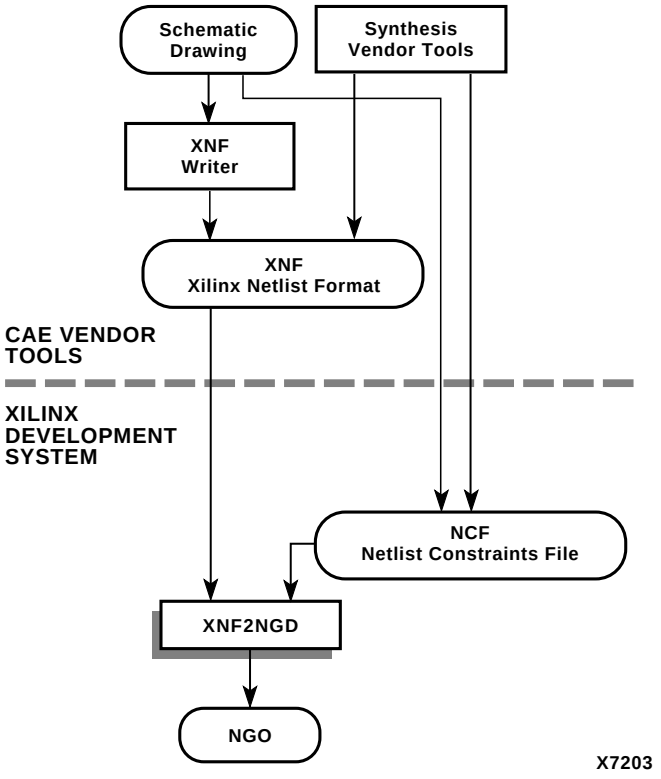


Figure B-2 XNF2NGD Design Flow

You can run XNF2NGD in the following ways.

- From the Design Manager / Flow Engine with the Translate step
- Automatically from NGDBuild
- From the UNIX or DOS command line, as described in the following sections.

Note When creating nets or symbols names, do not use reserved names. Reserved names are the names of symbols for primitives and macros in the *Libraries Guide* and net names GSR,RESET, GR, and PRELOAD. If you use these names, XNF2NGD issues an error.

XNF2NGD Syntax

The following command reads your XNF netlist and converts it to an NGO file.

xnf2ngd [*options*] *xnf_file ngo_file*

options can be any number of the XNF2NGD options listed in the "XNF2NGD Options" section. They do not need to be listed in any particular order. Separate multiple options with spaces.

xnf_file is the input file (in XNF format) to be converted. The file can have any extensions (for example, .xnf, .xtf, .xff, .xg, or .sxnf), as long as the file is in XNF format. If you enter a file name with no extension, XNF2NGD looks for a file with an .xnf extension and the name you specified.

ngo_file is the output file in NGO format. The output file name, its extension, and its location are determined in the following way.

- If you do not specify an output file name, the output file has the same name as the input file, with an .ngo extension.
- If you specify an output file name with no extension, XNF2NGD appends the .ngo extension to the file name.
- If you specify a file name with an extension other than .ngo, you get an error message and XNF2NGD does not run.
- If you do not specify a full path name, the output file is placed in the directory from which you ran XNF2NGD.

If the output file already exists, it is overwritten with the new file.

XNF2NGD Files

This section describes the XNF2NGD input and output files.

Input Files

XNF2NGD uses the following files as inputs.

- XNF file—This is the Xilinx Netlist Format (XNF) text file. The file can have any extension as long as the contents are in XNF format.
- NCF file—This Netlist Constraints File is produced by a vendor toolset and contains constraints specified within the toolset.

XNF2NGD reads the constraints in this file and adds the constraints to the output NGO file.

XNF2NGD reads the constraints in the NCF file if the NCF file has the same name as the input XNF file and an extension of .ncf. The name of the NCF file does not have to be entered on the XNF2NGD command line.

Output Files

The output of XNF2NGD is an NGO file—a binary file containing a logical description of the design in terms of its original components and hierarchy.

XNF2NGD Options

This section describes the XNF2NGD command options.

–f (Execute Commands File)

-f *command_file*

The –f option executes the command line arguments in the specified *command_file*. For more information on the –f option, see the "–f Option" section of the "Introduction" chapter.

–l (Libraries to Search)

-l *libname*

The –l option indicates the list of libraries to search when determining what library components were used to build the design. This information is necessary for NGDBuild, which must determine the source of the design's components before it can resolve the components to Xilinx primitives.

You can specify multiple –l options on the command line. Each must be preceded with –l; you cannot combine multiple *libname* specifiers after one -l. For example, **–l xilinxun synopsys** is not acceptable, while **–l xilinxun –l synopsys** is acceptable.

The allowable entries for *libname* are the following.

- **xilinxun** (For Xilinx Unified library)
- **synopsys**

- **XC3000**
- **XC4000**
- **XC9500**

Note You do not have to enter xilinxun with a –l option. The Xilinx Development System tools automatically access these libraries. In most cases, you do not have to enter XC3000 or XC4000 with a –l option. However, if your XNF file contains an input latch (INLAT) component and no part type is specified in the XNF file, the meaning of the INLAT component is ambiguous. In this case, XNF2NGD will stop with an error message. You must run XNF2NGD again using the –l option to define the INLAT component; –l XC3000 means the INLAT is transparent High and –l XC4000 means it is transparent Low.

–p (Part Name)

-p *part*

The –p option specifies the part into which your design is implemented. The –p option can specify an architecture only, a complete part specification (device, package, and speed), or a partial specification (for example, device and package only).

The syntax for the –p option is described in the "Part Numbers in Commands" section of the "Introduction" chapter. Examples of *part* entries are **XCV50-TQ144** and **XCV50-TQ144-5**.

If you do not specify a part when you run XNF2NGD, you will have to specify one when you run NGDBuild.

You may also use the –p option to override a part name in the input XNF netlist or a part name in an NCF file.

–r (Ignore LOC Properties)

The –r option filters out all location properties (LOC=) from the design. This can be used when you are migrating to a different device or architecture, because locations in one architecture do not match locations in another.

–u (Top-Level XNF Netlist)

The –u option specifies that the input XNF file is the top-level design netlist. When XNF2NGD translates netlists at lower hierarchical levels, XNF2NGD adds to the lower-level NGO file information that is unnecessary in the top-level NGO file. The –u option prevents this information from being added to the top-level NGO file.

NGDBuild

This program is compatible with the following families.

- Spartan / XL / -II
- Virtex / -E / -II
- XC9500 / XL / XV
- XC4000E / L / EX / XL / XV / XLA
- XC3000A / L
- XC3100A / L
- XC5200

NGDBuild performs all the steps necessary to read a netlist file in XNF or EDIF format and create an NGD file describing the logical design. The NGD file resulting from an NGDBuild run contains both a logical description of the design reduced to NGD primitives and a description in terms of the original hierarchy expressed in the input netlist. The output NGD file can be mapped to the desired device family.

Converting a Netlist to an NGD File

NGDBuild performs the following steps to convert a netlist to an NGD file. This flow is shown in the following figure.

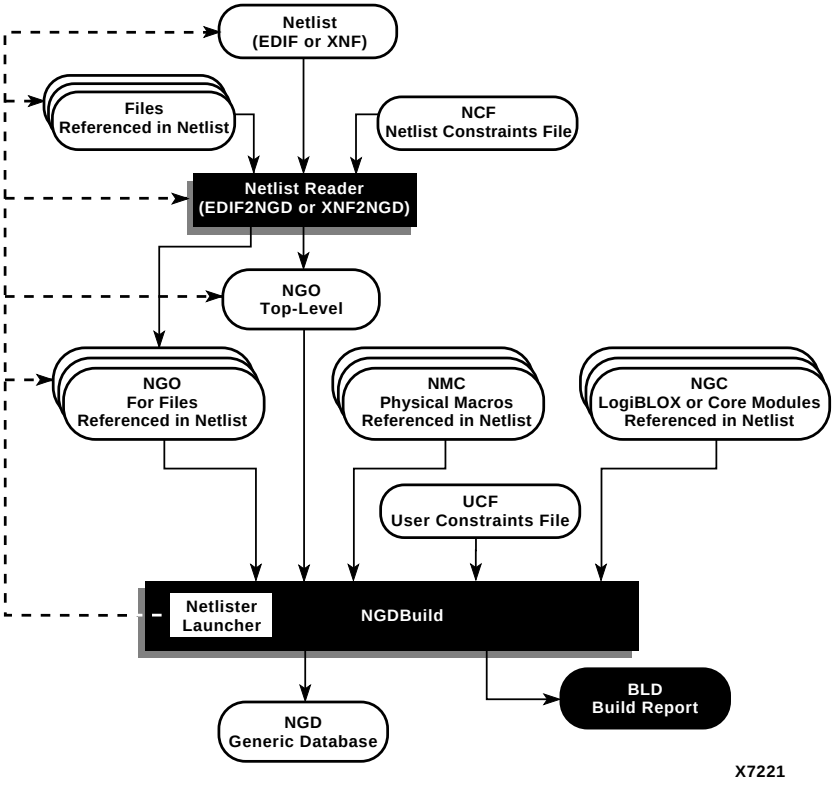


Figure B-3 NGDBuild and the Netlist Readers

1. Reads the source netlist

   To perform this step, NGDBuild invokes the Netlister Launcher, a part of the NGDBuild software which determines the type of the input netlist and starts the appropriate netlist reader program. If the input netlist is in EDIF or XNF format, the Netlister Launcher invokes EDIF2NGD or XNF2NGD. If the input netlist is in another format that the Netlister Launcher recognizes, the Netlister Launcher invokes the program necessary to convert the

netlist to EDIF or XNF format, then invokes EDIF2NGD or XNF2NGD. The netlist reader produces an NGO file for the top-level netlist file.

If any subfiles are referenced in the top-level netlist (for example, a PAL description file, or another schematic file), the Netlister Launcher invokes the appropriate netlist reader for each of these files to convert each referenced file to an NGO file.

The Netlister Launcher is described in the "Netlister Launcher" section. The netlist reader programs are described in the "File Names and Locations" section and the "XNF2NGD" section.

2. Reduces all components in the design to NGD primitives

   To perform this step, NGDBuild merges components that reference other files by finding the referenced NGO files. NGDBuild also finds the appropriate system library components, physical macros (NMC files) and behavioral models.

3. Checks the design by running a Logical DRC (Design Rule Check) on the converted design

   The Logical DRC is a series of tests on the logical design. It is described in "Logical Design Rule Check" chapter.

4. Writes an NGD file as output

When NGDBuild reads the source netlist, it detects any files or parts of the design that have changed since the last run of NGDBuild. It updates files as follows.

- If you have modified your input design since you last ran NGDBuild, NGDBuild updates all of the files affected by the change and use the updated files to produce a new NGD file.

  The Netlister Launcher checks timestamps (date and time information) for netlist files and intermediate NGDBuild files (NGOs). If an NGO file has a timestamp earlier than the netlist file that produced it, the NGO file is updated and a new NGD file is produced.

- NGDBuild completes the NGD production if all or some of the intermediate files already exist. These files may exist if you ran a netlist reader before you ran NGDBuild. NGDBuild uses the existing files and create the remaining files necessary to produce the output NGD file.

Note If the NGO for an netlist file is up to date, NGDBuild looks for an NCF file with the same base name as the netlist in the netlist directory and compares the timestamp of the NCF file against that of the NGO file. If the NCF file is newer, XNF2NGD or EDIF2NGD is run again. However, if an NCF file existed on a previous run of NGDBuild and the NCF file was deleted, NGDBuild will not detect that XNF2NGD or EDIF2NGD must be run again. In this case, you must use the –nt on option to force a rebuild.

Syntax, files, and options for the NGDBuild command are described in the "NGDBuild" chapter.

Bus Matching

When NGDBuild encounters an instance of one netlist within another netlist, it requires that each pin specified on the upper-level instance match to a pin (or port) on the lower-level netlist. Two pins must have exactly the same name in order to be matched. This requirement applies to all FPGAs and CPLDs supported for NGDBuild.

If the interface between the two netlists uses bused pins, these pins are expanded into scalar pins before any pin matching occurs. For example, the pin A[7:0] might be expanded into 8 pins namedA[7] through A[0]. If both netlists use the same nomenclature (that is, the same index delimiter characters) when expanding the bused pin, the scalar pin names will match exactly. However, if the two netlists were created by different vendors and different delimiters are used, the resulting scalar pin names do not match exactly.

In cases where the scalar pin names do not match exactly, NGDBuild analyzes the pin names in both netlists and attempts to identify names that resulted from the expansion of bused pins. When it identifies a bus-expanded pin name, it tries several other bus-naming conventions to find a match in the other netlist so it can merge the two netlists. For example, if it finds a pin named A(3) in one netlist, it looks for pins named A(3), A[3], A<3> or A3 in the other netlist.

Following are the bus naming conventions understood by NGDBuild.

Table B-1 Bus Naming Conventions

Naming Convention	Example
busname(index)	DI(3)
busname<index>	DI<3>
busname[index]	DI[3]
busnameindex	DI3

If your third-party netlist writer allows you to specify the bus-naming convention, use one of the conventions shown in the preceding table to avoid "pin mismatch" errors during NGDBuild. If your third-party EDIF writer preserves bus pins using the EDIF "array" construct, the bus pins will be expanded by EDIF2NGD using parentheses, which is one of the supported naming conventions.

Note NGDBuild support for bused pins is limited to this understanding of different naming conventions. It is not able to merge together two netlists if a bused pin has different indices between the two files. For example, it cannot match A[7:0] in one netlist to A[15:8] in another.

In the Xilinx UnifiedPro library for Virtex, some of the pins on the block RAM primitives are bused. If your third-party netlist writer uses one of the bus naming conventions listed in the preceding table or uses the EDIF array construct, these primitives are recognized properly by NGDBuild. The use of any other naming convention may result in an "unexpanded block" error during NGDBuild.

Netlister Launcher

The Netlister Launcher, which is part of NGDBuild, performs any netlist translations necessary to execute the NGDBuild command.

When NGDBuild is invoked, the Netlister launcher goes through the following steps.

1. The Netlister Launcher initializes itself with a set of rules for determining what netlist reader to use with each type of netlist, and the options with which each reader is invoked.).

The rules are contained in the system rules file (described in the "System Rules File" section) and in the user rules file (described in the "User Rules File" section).

2. NGDBuild makes the directory of the top-level netlist the first entry in the Netlister Launcher's list of search paths.
3. For the top-level design and for each file referenced in the top-level design, NGDBuild queries the Netlist Launcher for the presence of the corresponding NGO file.
4. For each NGO file requested, the Netlister Launcher performs the following actions.

   - Determines what netlist is the source for the requested NGO file

     The Netlister Launcher determines the source netlist by looking in its rules database for the list of legal netlist extensions. Then, it looks in the search path (which includes the current directory) for a netlist file possessing a legal extension and the same name as the requested NGO file.

   - Finds the requested NGO file

     The Netlister Launcher looks first in the directory specified with the –dd option (or current directory if a directory is not specified). If the NGO file is not found there and the source netlist was not found in the search path, the Netlister Launcher looks for the NGO file in the search path.

   - Determines whether the NGO file must be created or updated

     If neither the netlist source file nor the NGO file is found, NGDBuild exits with an error.

     If the netlist source file is found but the corresponding NGO file is not found, the Netlister Launcher invokes the proper netlist reader to create the NGO file.

     If the netlist source file is not found but the corresponding NGO file is found, the Netlister Launcher indicates to NGDBuild that the file exists and NGDBuild uses this NGO file.

     If both the netlist source file and the corresponding NGO file are found, the netlist file's time stamp is checked against the

NGO file's timestamp. If the timestamp of the NGO file is later than the source netlist, the Netlister Launcher returns a "found" status to NGDBuild. If the timestamp of the NGO file is earlier than the netlist source, or the NGO file is not present in the expected location, then the Launcher creates the NGO file from the netlist source by invoking the netlist reader specified by its rules.

Note The timestamp check can be overridden by options on the NGDBuild command line. The –nt on option updates all existing NGO files, regardless of their timestamps. The –nt off option does not update any existing NGO files, regardless of their timestamps.

5. The Netlister launcher indicates to NGDBuild that the requested NGO files have been found, and NGDBuild can process all of these NGO files.

Netlister Launcher Rules Files

The behavior of the Netlister Launcher is determined by rules defined in the system rules file and the user rule file. These rules determine the following.

- What netlist source files are acceptable
- Which netlist reader reads each of these netlist files
- What the default options are for each netlist reader

The system rules file contains the default rules supplied with the Xilinx Development System software. The user rules file can add to or override the system rules.

The following sections describe the user rules file and the system rules.

User Rules File

The user rules file can add to or override the rules in the system rules file. You can specify the location of the user rules file with the –ur option to the NGDBuild command line. The user rules file must have a .urf extension. See the "–ur (Read User Rules File)" section of the "NGDBuild" chapter for more information.

User Rules and System Rules

User rules are treated as described below.

- A user rule can override a system rule if it specifies the same source and target files as the system rule.
- A user rule can supplement a system rule if its target file is identical to a system rule's source file, or if its source file is the same as a system rule's target file.
- A user rule that has a source file identical to a system rule's target file and a target file that is identical to the same system rule's source file is illegal, because it defines a loop.

User Rules Format

Each rule in the user rules file has the following format.

```
RuleName = <rulename1>;

<key1> = <value1>;

<key2> = <value2>;

  .

  .

  .

<keyn> = <valuen>;
```

The following is a description of the keys allowed and the values expected.

Note All of the values mentioned in the following paragraphs are described in the "Value Types in Key Statements" section.

- RuleName—This key identifies the beginning of a rule. It is also used in error messages relating to the rule. It expects a RULENAME value. A value is required.
- NetlistFile—This key specifies a netlist or class of netlists that the netlist reader takes as input. The extension of NetlistFile is used

together with the TargetExtension to identify the rule. It expects either a FILENAME or an EXTENSION value. If a file name is specified, it should be just a file name (that is, no path). Any leading path is ignored. A value is required.

- TargetExtension—This key specifies the class of files generated by the netlist reader. It is used together with the extension from NetlistFile to identify the rule. It expects an EXTENSION value. A value is required.
- Netlister—This key specifies the netlist reader to use when translating a specific netlist or class of netlists to a target file. The specific netlist or class of netlists is specified by NetlistFile, and the class of target files is specified by TargetExtension. It expects an EXECUTABLE value. A value is required.
- NetlisterTopOptions—This key specifies options for the netlist reader when compiling the top level design. It expects an OPTIONS value or the keyword NONE. Included in this string should be the keywords $INFILE and $OUTFILE, in which the input and output files is substituted. In addition, the following keywords may appear.
  - $PART—The part passed to NGDBuild by the –p switch is substituted. It may include architecture, device, package and speed information. The syntax for a $PART specification is the same as described in the “Part Numbers in Commands” section of the “Introduction” chapter.
  - $FAMILY—The family passed to NGDBuild by the –p switch is substituted. A value is optional.
  - $DEVICE—The device passed to NGDBuild by the –p switch is substituted. A value is optional.
  - $PKG—The package passed to NGDBuild by the –p switch is substituted. A value is optional.
  - $SPEED—The speed passed to NGDBuild by the –p switch is substituted. A value is optional.
  - $LIBRARIES—The libraries passed to NGDBuild. A value is optional.
  - $IGNORE_LOCS—Substitute the –r option to EDIF2NGD or XNF2NGD if the NGDBuild command line contained a –r option.

- $ADD_PADS—Substitute the –a option to EDIF2NGD if the NGDBuild command line contained a –a option.

The options in the NetlisterTopOptions line must be enclosed in quotation marks.

- NetlisterOptions—This key specifies options for the netlist reader when compiling sub-designs. It expects an OPTIONS value or the keyword NONE. Included in this string should be the keywords $INFILE and $OUTFILE, in which the input and output files is substituted. In addition, any of the keywords that may be entered for the NetlisterTopOptions key may also be used for the NetlisterOptions key.

  The options in the NetlisterOptions line must be enclosed in quotation marks.

- NetlisterDirectory—This key specifies the directory in which to run the netlist reader. The launcher changes to this directory before running the netlist reader. It expects a DIR value or the keywords $SOURCE, $OUTPUT, or NONE, where the path to the source netlist is substituted for $SOURCE, the directory specified with the -dd option is substituted for $OUTPUT, and the current working directory is substituted for NONE. A value is optional.

- NetlisterSuccessStatus—This key specifies the return code that the netlist reader returns if it ran successfully. It expects a NUMBER value or the keyword NONE. The number may be preceded with one of the following: =, <, >, or !. A value is optional.

Value Types in Key Statements

The value types used in the preceding key statements are the following.

- RULENAME—Any series of characters except for a semicolon (;) and white space (for example, space, tab, newline).
- EXTENSION—A "." followed by an extension that conforms to the requirements of the platform.
- FILENAME—A file name that conforms to the requirements of the platform.

- EXECUTABLE—An executable name that conforms to the requirements of the platform. It may be a full path to an executable or just an executable name. If it is just a name, then the $PATH environment variable is used to locate the executable.
- DIR—A directory name that conforms to the requirements of the platform.
- OPTIONS—Any valid string of options for the executable.
- NUMBER—Any series of digits.
- STRING—Any series of characters in double quotes.

System Rules File

The system rules are shown following. The system rules file is not an ASCII file, but for the purpose of describing the rules, the rules are described using the same syntax as in the user rules file. This syntax is described in the "User Rules File" section.

Note If a rule attribute is not specified, it is assumed to have the value NONE.

```
#########################################################
# xnf2ngd rules
#########################################################

RuleName = XNF_RULE;
NetlistFile = .xnf;
TargetExtension = .ngo;
Netlister = xnf2ngd;
NetlisterTopOptions = "[-p $PART] -u [$IGNORE_LOCS] {-l $LIBRARIES}
$INFILE $OUTFILE";
NetlisterOptions = "[-p $PART] [$IGNORE_LOCS] {-l $LIBRARIES}
$INFILE $OUTFILE";
NetlisterDirectory = NONE;
NetlisterSuccessStatus = 0;

RuleName = XTF_RULE;
NetlistFile = .xtf;
TargetExtension = .ngo;
Netlister = xnf2ngd;
NetlisterTopOptions = "[-p $PART] -u [$IGNORE_LOCS] {-l $LIBRARIES}
$INFILE $OUTFILE";
```

```
NetlisterOptions = "[-p $PART] [$IGNORE_LOCS] {-l $LIBRARIES}
$INFILE $OUTFILE";
NetlisterDirectory = NONE;
NetlisterSuccessStatus = 0;

RuleName = XFF_RULE;
NetlistFile = .xff;
TargetExtension = .ngo;
Netlister = xnf2ngd;
NetlisterTopOptions = "[-p $PART] -u [$IGNORE_LOCS] {-l $LIBRARIES}
$INFILE $OUTFILE";
NetlisterOptions = "[-p $PART] [$IGNORE_LOCS] {-l $LIBRARIES}
$INFILE $OUTFILE";
NetlisterDirectory = NONE;
NetlisterSuccessStatus = 0;

RuleName = XG_RULE;
NetlistFile = .xg;
TargetExtension = .ngo;
Netlister = xnf2ngd;
NetlisterTopOptions = "[-p $PART] -u [$IGNORE_LOCS] {-l $LIBRARIES}
$INFILE $OUTFILE";
NetlisterOptions = "[-p $PART] [$IGNORE_LOCS] {-l $LIBRARIES}
$INFILE $OUTFILE";
NetlisterDirectory = NONE;
NetlisterSuccessStatus = 0;

RuleName = SYN_XNF_RULE;
NetlistFile = .sxnf;
TargetExtension = .ngo;
Netlister = xnf2ngd;
NetlisterTopOptions = "[-p $PART] -u [$IGNORE_LOCS] -l synopsys {-l
$LIBRARIES} $INFILE $OUTFILE";
NetlisterOptions = "[-p $PART] [$IGNORE_LOCS] -l synopsys {-l
$LIBRARIES}
$INFILE $OUTFILE";
NetlisterDirectory = NONE;
NetlisterSuccessStatus = 0;

#######################################################
# edif2ngd rules
```

```
#########################################################

RuleName = EDN_RULE;
NetlistFile = .edn;
TargetExtension = .ngo;
Netlister = edif2ngd;
NetlisterTopOptions = "[$IGNORE_LOCS] [$ADD_PADS] {-l $LIBRARIES}
$INFILE $OUTFILE";
NetlisterOptions = "-noa [$IGNORE_LOCS] {-l $LIBRARIES} $INFILE
$OUTFILE";
NetlisterDirectory = NONE;
NetlisterSuccessStatus = 0;

RuleName = EDF_RULE;
NetlistFile = .edf;
TargetExtension = .ngo;
Netlister = edif2ngd;
NetlisterTopOptions = "[$IGNORE_LOCS] [$ADD_PADS] {-l $LIBRARIES}
$INFILE $OUTFILE";
NetlisterOptions = "-noa [$IGNORE_LOCS] {-l $LIBRARIES} $INFILE
$OUTFILE";
NetlisterDirectory = NONE;
NetlisterSuccessStatus = 0;

RuleName = EDIF_RULE;
NetlistFile = .edif;
TargetExtension = .ngo;
Netlister = edif2ngd;
NetlisterTopOptions = "[$IGNORE_LOCS] [$ADD_PADS] {-l $LIBRARIES}
$INFILE $OUTFILE";
NetlisterOptions = "-noa [$IGNORE_LOCS] {-l $LIBRARIES} $INFILE
$OUTFILE";
NetlisterDirectory = NONE;
NetlisterSuccessStatus = 0;

RuleName = SYN_EDIF_RULE;
NetlistFile = .sedif;
TargetExtension = .ngo;
Netlister = edif2ngd;
NetlisterTopOptions = NONE;
NetlisterOptions = "-l synopsys [$IGNORE_LOCS] {-l $LIBRARIES}
```

```
$INFILE $OUTFILE";
NetlisterDirectory = NONE;
NetlisterSuccessStatus = 0;

#######################################################
# other rules
#######################################################

RuleName = PLD_RULE;
NetlistFile = .pld;
TargetExtension = .xnf;
Netlister = readpld;
NetlisterTopOptions = "-f $INFILE -t -ox $OUTFILE";
NetlisterOptions = "-f $INFILE -ox $OUTFILE";
NetlisterDirectory = NONE;
NetlisterSuccessStatus = 0;
```

Rules File Examples

The following sections provide examples of system and user rules. The first example is the basis for understanding the ensuing user rules examples.

Example 1: EDF_RULE System Rule

As shown in the “System Rules File” section, the EDF_RULE system rule is defined as follows.

```
RuleName = EDF_RULE;
NetlistFile = .edf;
TargetExtension = .ngo;
Netlister = edif2ngd;
NetlisterTopOptions = "[$IGNORE_LOCS] [$ADD_PADS] {-l $LIBRARIES}
$INFILE $OUTFILE";
NetlisterOptions = "-noa [$IGNORE_LOCS] {-l $LIBRARIES} $INFILE
$OUTFILE";
NetlisterDirectory = NONE;
NetlisterSuccessStatus = 0;
```

The EDF_RULE instructs the Netlister Launcher to use EDIF2NGD to translate an EDIF file to an NGO file. If the top-level netlist is being translated, the options defined in NetlisterTopOptions are used; if a lower-level netlist is being processed, the options defined by NetlisterOptions are used. Because NetlisterDirectory is NONE, the Netlister Launcher runs EDIF2NGD in the current working directory (the one from which NGDBuild was launched). The launcher expects EDIF2NGD to issue a return code of 0 if it was successful; any other value is interpreted as failure.

Example 2: User Rule

Following is a another example of a User Rule.

```
// URF Example 2
RuleName = OTHER_RULE; // end-of-line comments are also allowed
NetlistFile = .oth;
TargetExtension = .edf;
Netlister = other2edf;
NetlisterOptions = "$INFILE $OUTFILE";
NetlisterSuccessStatus = 1;
```

The user rule OTHER_RULE defines a completely new translation, from a hypothetical OTH file to an EDIF file. To do this translation, the other2edf program is used. The options defined by NetlisterOptions are used for translating all OTH files, regardless of whether they are top-level or lower-level netlists (because no explicit NetlisterTopOptions is given). The launcher expects other2edf to issue a return code of 1 if it was successful; any other value be interpreted as failure.

After the Netlister Launcher has used OTHER_RULE to run other2edf and create an EDIF file, it uses the EDF_RULE system rule (shown in the preceding section) to translate the EDIF file to an NGO file.

Example 3: User Rule

Following is a another example of a User Rule.

```
// URF Example 3
RuleName = EDF_LIB_RULE;
NetlistFile = .edf;
TargetExtension = .ngo;
NetlisterOptions = "-l xilinxun $INFILE $OUTFILE";
```

Because both the NetlistFile and TargetExtension of this user rule match those of the system rule EDF_RULE (shown in the "Example 1: EDF_RULE System Rule" section), the EDF_LIB_RULE overrides the EDF_RULE system rule. Any settings that are not defined by the EDF_LIB_RULE are inherited from EDF_RULE. So EDF_LIB_RULE uses the same netlister (EDIF2NGD), the same top-level options, the same directory, and expects the same success status as EDF_RULE. However, when translating lower-level netlists, the options used are only "–l xilinxun $INFILE $OUTFILE." (There is no reason to use "–l xilinxun" on EDIF2NGD; this is for illustrative purposes only.)

Example 4: User Rule

Following is a another example of a User Rule.

```
// URF Example 4
RuleName = STATE_EDF_RULE;
NetlistFile = state.edf;
TargetExtension = .ngo;
Netlister = state2ngd;
```

Although the NetlistFile is a complete file name, this user rule also matches the system rule EDF_RULE (shown in the "Example 1: EDF_RULE System Rule" section), because the extensions of Netlist-File and TargetExtension match. When the Netlister Launcher tries to make a file called state.ngo, it uses this rule instead of the system rule EDF_RULE (assuming that state.edf exists). As with the previous example, the unspecified settings are inherited from the matching system rule. The only change is that the fictitious program state2ngd is used in place of EDIF2NGD.

Note that if EDF_LIB_RULE (from the example in the "Example 3: User Rule" section) and this rule were both in the user rules file, STATE_EDF_RULE includes the modifications made by EDF_LIB_RULE. So a lower-level state.edf is translated by running state2ngd with the "-l xilinxun" option.

File Names and Locations

Following are some notes about file names and notations in NGDBuild.

- An intermediate file has the same root name as the design that produced it. An intermediate file is generated when more than one netlist reader is needed to translate a netlist to a NGO file.
- Netlist root file names in the search path must be unique. For example, if you have the design state.edn, you cannot have another design named state.xnf in any of the directories specified in the search path.
- NGDBuild and the Netlister Launcher support quoted file names. Quoted file names may have special characters (for example, a space) that are not normally allowed.
- If the output directory specified in the call to NGDBuild is not writable, an error is displayed and NGDBuild fails.

Index

Numerics

A

B

C

D

E

F

G

H

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X