Chapter 4

# Cadence Verilog-XL Interface and Libraries

This chapter contains the following information on using the Xilinx Interface to Cadence Verilog-XL and the Cadence Verilog-XL Libraries.

- Introduction
- Contents
- Other Cadence Interface Products
- Other Sources of Information
- Installation
- Features in This Release
    - X2VPrep Program
    - XNF2Verilog Program
- Simulation Scripts
    - funcnetx
    - timenetx
- Detailed Design Flow for Design Verification
- Known Issues[1]

1. For information about Core Tools known issues, refer to the XACTstep Version 5.2 / 6.0 Core Tools Known Issues, Work Arounds, and Helpful Hints.

## Introduction

Welcome to the Cadence Verilog-XL Interface and Libraries Package from Xilinx!

The Xilinx Interface to Cadence Verilog-XL, ES-Verilog, is supported on HP-700 and Sun4 workstations only. It is released as ES (Engineering Sample) software to accelerate the product's availability. In a future release, the Verilog libraries and interface will become a part of the FPGA Core Implementation Tools (DS-502) on HP700 and Sun4 platforms as well as HP700 and Sun4 Standard packages.

**Note:** For Synopsys users, this release only supports post-route timing simulation. Pre-route, post-synthesis, functional simulation of Synopsys designs is not supported.

**Note:** Functional simulation of designs implemented using the pre-Unified libraries is not supported by this package. If your design is entered using pre-Unified libraries, you must use the Cadence 9402 interface to process it. The application note "Using the Xilinx Interface 4.0 with XACT 5.0" describes the procedure for processing pre-Unified designs in detail. Contact Cadence Technical Support at 1-800-223-3622 or Xilinx Technical Support at 1-800-255-7778 to request a copy.

## Contents

The Development System (DS) product you received contains software, documentation, and/or hardware. New DS Base, Standard, and Extended packages contain hardware, software, and documentation. Interface and Update products have software and documentation only.

### Software

The ES Release Version 5.2.0 Verilog-XL Interface and Libraries (ES-Verilog) are available only on CD-ROM, and includes the Verilog interface for both Sun4 series workstations and HP700 workstations.

### Documentation

This release document is the only document necessary to use the Verilog-XL Interface. However, it also lists other references that you may find helpful.

### Maintenance and Support

This product comes with free technical and product information telephone support (toll-free in the U.S. and Canada). You can also fax and e-mail your questions. See the last page of this release note for offices and phone numbers.

Xilinx also offers many customer education services; ask your sales representative for more information.

## Other Cadence Interface Products

The following products are also available from Xilinx.

### DS-381-SN2-C (Sun4) and DS-381-HP7-C (HP700)

These are for users who purchase DS-502 and / or DS-550 separately.

- Concept libraries and interface
- Composer libraries and interface
- Verilog simulation models and interface
- Rapidsim simulation models and interface

### DS-CDN-SN2-C (Sun4) and DS-CDN-HP7-C (HP700)

These are complete front-to-back solutions.

- All software included in DS-381, plus:
- FPGA Core Implementation Tools (DS-502)
- XEPLD Translator Core Tools (DS-550)
- X-BLOX Synthesis Tool (DS-380)

The two sets of products above include some features that are not included in the Xilinx Interface to Verilog-XL:

- Two scripts, timenet and funcnet, which automate the preparation of designs for simulation
- Two scripts, timesim and funcsim, which automate the simulation process
- Online documentation

- Support for Cadence Rapidsim, Composer, and Concept integration and back-annotation
- Support for Cadence Logic Workbench integration
- Support for Cadence System Workbench integration

## Other Sources of Information

Additional sources of information on the Xilinx Interface to Verilog-XL and related products are listed in the next two sections.

### Related Manuals

The Xilinx Interface to Verilog-XL is used with other Cadence and Xilinx products during the Xilinx FPGA and EPLD design process. The following manuals give you more information about the tools used with the Xilinx Interface to Verilog-XL.

| Manual | What It Contains |
|---|---|
| *Verilog-XL Configuration Guide*<br>*Verilog-XL Reference Manual*<br>*Verilog-XL Tutorial* | Information on the Verilog-XL simulator |
| *Libraries Guide* | Information on the primitives and macros available in the XACT libraries |
| *Hardware & Peripherals User Guide* | Information about the Xilinx demonstration boards, the XChecker / download cable, the Xilinx HW112 PROM programmer, the XPP PROM Programmer software interface, and the XChecker software interface |
| *Development System Reference Guide, Vols 1-3* | Information about the programs within the XACT*step* Development System |
| *Development System User Guide* | Overview of the XACT tools and the Xilinx FPGA design process |
| *X-BLOX Reference/User Guide* | Information about the X-BLOX library and software |
| *XEPLD Design Guide* | Information about how to use the XEPLD software to create designs for Xilinx XC7000-series EPLD devices |

| Manual | What It Contains |
|---|---|
| *XEPLD Reference Guide* | Reference information about the XEPLD software used for implementing Xilinx XC7000-series EPLD devices |
| *The Programmable Logic Data Book* | Complete specifications for the Xilinx FPGA and EPLD devices |

## Installation

This section gives instructions for installing the Xilinx Interface to Verilog-XL.

### Software Requirements

The Xilinx Interface to Verilog-XL is provided on CD-ROM. To install the Xilinx Interface to Verilog-XL, you must have a CD-ROM drive and at least 4 MB of free disk space.

### Software Versions

The ES-Verilog Version 5.2.0 Xilinx Interface to Verilog-XL includes the following software:

| Program | Version |
|---|---|
| X2VPrep | 9504 |
| XNF2Verilog | 9504 |
| funcnetx | 5.4.1c |
| timenetx | 5.2.0 |

### Installing from CD-ROM

You can install the Xilinx Interface to Verilog-XL from CD-ROM on the following workstation types.

- Sun4 (SPARC) with SunOS 4.1.x or SunOS 5.3
- HP-PA Model 9000 (RISC) with HP-UX 9.01

Follow these procedures.

1. Check the system requirements defined in the *Getting Started & Installation Guide.*
2. Insert the CD-ROM into the drive.
3. Start a window manager (OpenWindows or X-Windows).
4. Execute the platform-specific commands as indicated in the following sections.

   On A Sun4 Workstation, make these entries:

   ```
   # mkdir /cdrom
   # mount -t hsfs -o ro /dev/sr0 /cdrom
   # cd xilinx_install_dir
   ```

   (to change directory to your XACT install directory)

   ```
   # uudecode /cdrom/xbbs/vlog_intfc/
   xil_vlog_intfc.sun4.tar.z.uu
   # uncompress vlog_intfc.sun4.tar.z
   # tar xvf vlog_intfc.sun4.tar
   ```

   On an HP700 Workstation, make these entries:

   ```
   # mkdir /cdrom
   # mount /dev/dsk/3s0 /cdrom
   # cd xilinx_install_dir
   ```

   (to change directory to your XACT install directory)

   ```
   # uudecode /cdrom/xbbs/vlog_intfc/
   xil_vlog_intfc.hp7.tar.z.uu
   # uncompress xil_vlog_intfc.hp7.tar.z
   # tar xvf xil_vlog_intfc.hp7.tar
   ```

**Note:** Workstation users must have root privileges to use mount commands (“#” is the root prompt, and “%” is the user prompt). Directories and device names can vary; therefore, check these names with your system administrator.

5. Xilinx recommends that you use your XACT directory as the install directory. With XACT as the install directory, the installation will create the following directory structure for you:

   ```
   $XACT/bin/platform/ (where “platform” is either sparc
   or hppa)
   ```

```
x2vprep
xnf2verilog
funcnetx
timenetx

$XACT/data/ (Pin files)
          xc2000.pin
          xc3000.pin
          xc4000.pin
          xc7000.pin

$XACT/ (Verilog libraries)
     verilog2000/
     verilog3000/
     verilog4000/
     verilog7000/
```

## Environment Variable Settings

Before using the Xilinx Interface to Verilog-XL, you need to add the name of your installation directory to the path variable in your .cshrc file, if different from your Xilinx installation directory:

```
set path = (install_dir/bin/platform $path)
```

where install_dir is your installation directory, and platform is either sparc, for a Sun4 platform, or hppa, for an HP700 platform.

Example:

```
set path = ( /tools/xact/bin/sparc $path )
```

In this example, the install directory is /tools/xact, and the platform is Sun4 (sparc).

To run the Xilinx FPGA Core Tools programs X-BLOX, LCA2XNF, and XNFBA, you will also need to:

1. Add the path to your Xilinx Development System Core Tools installation directory to your path variable, if different from your ES-Verilog installation directory:

   ```
   set path = (core_install_dir/bin/platform $path)
   ```

2. Define an environment variable, “XACT”, and set it to the name of the Core Tools installation directory.

   ```
   setenv XACT core_install_dir
   ```

In addition, Xilinx highly recommends that you define the environment variable, CDS_INTFC, in your .cshrc file or setup file. CDS_INTFC should be declared in either file as follows:

```
setenv CDS_INTFC install_dir
```

where install_dir is the installation directory for your Xilinx Interface to Verilog-XL.

Both funcnetx and timenetx will search for Cadence pin files and Verilog libraries under $CDS_INTFC, if it is defined.

Example:

```
setenv CDS_INTFC /tools/xact
```

In the example, the install directory is /tools/xact.

To run XEPLD Translator Core Tools executables (including vmh2xnf), you must also install the DS-550 software.

Refer to the *Getting Started & Installation Guide* for more details on setting up your environment for the Xilinx Development System.

### Licensing

The Xilinx Interface to Verilog-XL V5.2.0 executables, X2VPrep and XNF2Verilog, are unlicensed executables, so no license authorization is required to run these programs.

## Features in This Release

This section describes the supported devices, libraries, and programs in this release.

### Devices Supported

The Xilinx Interface to Verilog-XL enables you to simulate both Xilinx field-programmable gate array (FPGA) designs or erasable programmable logic device (EPLD) designs that are generated by the following products:

- XACT (Xilinx Automated CAE Tools) Development System, Version 5.0, 5.1.0, and 5.2.0.
- Xilinx XEPLD software, Version 5.0, 5.1.0, and 5.2.0.

The Xilinx Interface to Verilog-XL supports all released Xilinx FPGA families: XC2000, XC3000, XC4000, and XC7000 (the Xilinx EPLD family).

## Libraries

Included in the Xilinx Interface to Verilog-XL are the Verilog-XL simulation module libraries: verilog2000, verilog3000, verilog4000, and verilog7000.

## X2VPrep Program

X2VPrep is a Cadence executable that is run before the execution of XNF2Verilog. XNF2Verilog only handles a very specific and limited subset of the XNF syntax, and X2VPrep ensures that the XNF file used as input to XNF2Verilog uses only this subset of XNF.

**Note:** If the output file is not specified, the input file is overwritten after it is processed by X2VPrep.

X2VPrep adds EXT records for sourceless and loadless signals to the input XNF file when preparing a design sub-block containing X-BLOX modules for functional simulation. It also removes INV properties from non-combinatorial symbol pins, replacing them with INV symbols in the input XNF file in preparation for processing by XNF2Verilog. When processing a design containing X-BLOX modules for functional simulation, X2VPrep must be run both before XNFMerge and after X-BLOX.

### Syntax

To run X2VPrep, type the following syntax at the command line:

```
x2vprep input_file[.xnf|.xg] [output_file.[xnf|xcd]]
```

The parameters of this syntax are as follows:

- *input_file*[.xnf | .xg] is the XNF file that X2VPrep reads. If you do not provide a file extension, X2VPrep looks for a file with an .xnf extension.
- *output_file*.[xnf | xcd] is the name of the XNF file that X2VPrep generates. This parameter is optional; however, it is recommended that you do specify a name for the output file with a .xcd

extension to prevent X2VPrep from overwriting the original input file.

Example:

```
x2vprep **foo.xnf** foo.xcd
```

To display the help text for X2VPrep, type:

```
x2vprep -help
```

## XNF2Verilog Program

XNF2Verilog reads your XNF file and a pin file, and generates the files that you need to perform simulation with the Verilog-XL simulator. XNF2Verilog creates the following files from your XNF netlist and a pin file:

- A Verilog model (.v) file
- A stimulus template (.stim) file
- An SDF delay (.sdf) file

The XNF file must be a gate-level description. All net and pin names generated are Verilog-compliant.

### Syntax

To run XNF2Verilog, type the following syntax at the command line:

```
xnf2verilog input_file.[xnf|xcd|xg] output_file \
OPTIONS
```

where OPTIONS corresponds to the following options, all of which are required:

```
-arch architecture
```

```
-pin path_to_pinfile
```

```
-vlibs path_to_libraries
```

For timing simulation, there is an additional option to generate the Standard Delay Format File (SDF):

**-s** *sdf_file_name*

The parameters of this syntax are the following.

- *input_file*.[xnf | xcd | xg] is the XNF file that XNF2Verilog reads. If no file extension is provided, XNF2Verilog looks for a file with an .xnf extension.
- *output_file* (or Vlog Module Name) is the name to be used for the output .v and .stim files; this parameter is required.

  Example:

  ```
  xnf2verilog input_file.[xnf|xcd] output_file-arch
  architecture -pin path_to_pinfile -vlibs
  path_to_libraries
  ```

Option Descriptions

- -arch *architecture* is a required option. It instructs XNF2Verilog to use the package files for the family specified by architecture. Valid architecture values are 2000, 3000, 4000, and 7000.

  Example:

  ```
  -arch 7000
  ```

- -pin *path_to_pinfile* is a required option. It gives XNF2Verilog the name of the pin file, which must be specified by an explicit path starting from the root directory. The pin file name corresponds to the Xilinx architecture that you are targeting. Valid pin file names are xc2000.pin, xc3000.pin, xc4000.pin, or xc7000.pin for XC2000, XC3000, XC4000, or XC7000 devices, respectively. The pin files contain information about how to connect unconnected pins in the XNF file. It also contains information on whether a pin is sizable and whether it should have a delay. XNF2Verilog uses this information when generating the Verilog netlist. The pin files are normally located in the directory install_dir / data / and thus are specified as:

  `-pin` *install_dir*`/data/xc_family.pin`

  where *install_dir* is your Xilinx Interface to Verilog-XL installation directory, and xc_family.pin must be replaced with xc2000.pin, xc3000.pin, xc4000.pin, or xc7000.pin.

  Example:

  ```
  -pin /tools/xact/data/xc7000.pin
  ```

- -vlibs *path_to_libraries* is a required option. The Xilinx Interface to Verilog-XL includes a set of Verilog libraries. When XNF2Verilog runs, it creates a Verilog file containing a Uselib statement pointing to these libraries. This statement points Verilog-XL to the location of the simulation primitives when it runs a simulation. You must use the -vlibs option to point to the Verilog library directory appropriate to the architecture you are targeting. The default location of the Verilog libraries is the following:

*install_dir*/*verilog_family*

where *verilog_family* is either verilog2000, verilog3000, verilog4000, or verilog7000. For example, if *install_dir* is specified as an XACT directory called /tools/xact, the Verilog libraries for an XC4000 design will usually be located in /tools/xact/verilog4000.

Example:

```
-vlibs /tools/xact/verilog4000
```

- -s *sdf_filename*.sdf specifies the name of the .sdf delay annotation file to be generated by XNF2Verilog. This flag is optional. XNF2Verilog will not generate an .sdf file unless the -s option is specified. The .sdf file contains all of a design's timing information, so you must specify this option when preparing a design for timing simulation. The .sdf extension is recommended, but not necessary.

## Input Files

XNF2Verilog requires the following files:

- *input_file*.[xnf|xcd|xg] is the input XNF file, which may have an extension of .xnf, .xcd or .xg. XNF2Verilog accepts unrouted XNF files from XNFMerge, unrouted, synthesized netlists from X-BLOX, and routed XNF files from LCA2XNF. In all cases, the file must first be processed by X2VPrep. See the "Design Flow" section for more details.
- *pin_file*. See the above section on XNF2Verilog options under "-pin."

## Output Files

XNF2Verilog creates the following files:

- *output_file*.sdf contains the delay information to be annotated to the Verilog primitives. The .sdf file is only generated when you specify the -s option.

- *output_file*.stim is a template Verilog stimulus file, which contains an instantiation of the top-level Verilog module described in output_file.v. This file sets all inputs to 0 (zero) at initialization and includes a $gr_waves statement that includes all inputs and outputs in the top-level module.
- *output_file*.v is the Verilog netlist file, which you use as input to the Verilog-XL simulator. It contains instantiations of Xilinx primitives, whose Verilog models are in *install_dir* / *verilog_family*, where *verilog_family* is either verilog2000, verilog3000, verilog4000, or verilog7000. This file does not contain timing information; all timing information is stored in the SDF file.

  Example:

  ```
  xnf2verilog foo.xcd foo -s foo.sdf -arch 4000 \
  -pin /tools/xact/data/xc4000.pin -vlibs /tools\
  /xact/verilog4000
  ```

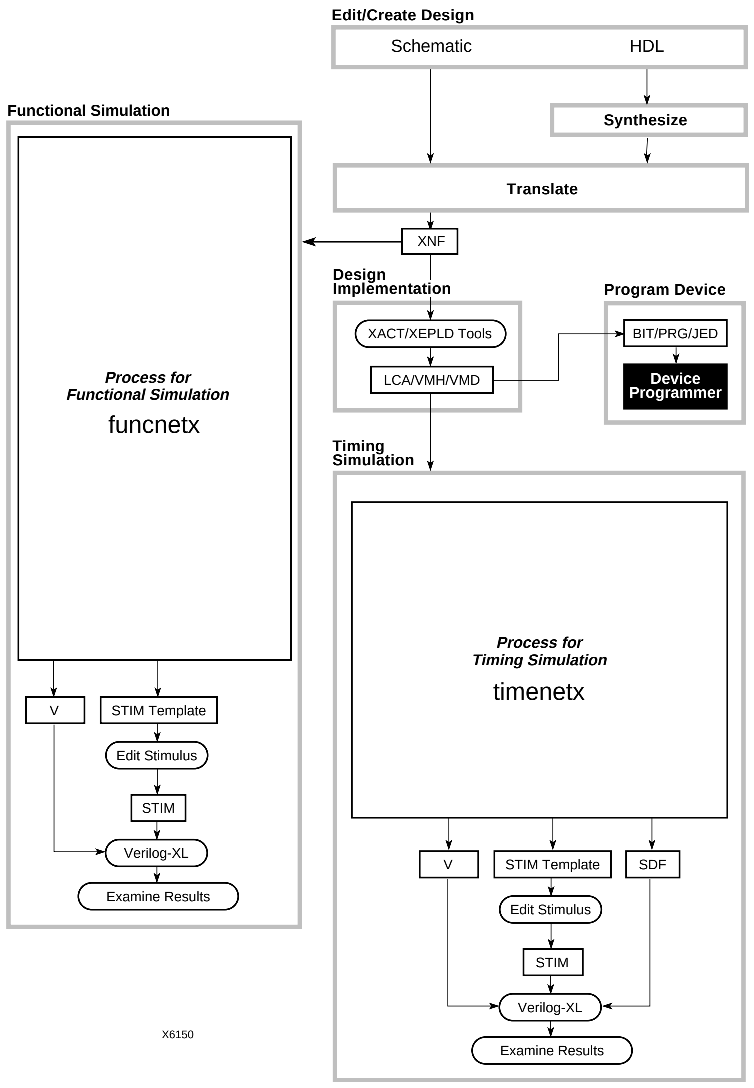
In this case, the design, foo, is an XC4000 netlist. The input file is foo.xcd, and the output files are foo.v, foo.sdf, and foo.stim. The XC4000 pin file specified in this example is located in /tools/xact/data, and the library being used is in /tools/xact/verilog4000.

## Simulation Scripts

This section provides reference information on the following Xilinx Interface to Verilog-XL scripts:

- funcnetx
- timenetx

Funcnetx and timenetx provide a convenient one-step means of generating a Verilog simulation netlist. Use funcnetx to process a design for functional simulation, and timenetx to process a design for timing simulation. The basic design flow is illustrated in the following figure.



## funcnetx Script

funcnetx is a C-shell script that runs x2vprep and xnf2verilog to create files for Verilog-XL functional simulation. For information about x2vprep and xnf2verilog, see the “x2vprep” and “xnf2verilog” sections in this document. The funcnetx script is based on the Cadence funcnet script, which is shipped with the DS381 interface.

If the environment variable, CDS_INTFC, is defined, the location of the pin file for xnf2verilog is assumed to be the following:

```
$CDS_INTFC/xcXXXXX.pin
```

and the location of the appropriate library will be assumed to be

```
$CDS_INTFC/verilogXXXX
```

where XXXX is either 2000, 3000, 4000, or 7000. If CDS_INTFC is not defined, you must specify the pin file and library locations explicitly using the -pin and -vlibs options.

## Syntax

funcnetx *design_name*
{2000 | 3000 | 4000 | 7000}
[-r *run_dir*] [-pin *pin_file]*
[-vlibs *path_to_libraries*]
[-noio] [-save]

The parameters of this syntax are as follows:

| | |
|---|---|
| *design_name* | (Required) flattened, gate-level XNF file. |
| 2000\|3000\|4000\|7000 | (Required) The target architecture family of your design. Funcnetx determines the appropriate flow to use based on the target architecture. |
| [ -r *run_dir* ] | (optional) Specifies the directory where *funcnetx* looks for the input Xilinx XNF files. The default is the current working directory. |
| [ -pin <pin_file> ] | (optional) Use specified pin file.This is not needed if the variable CDS_INTFC is declared in your environment. |

| | |
|---|---|
| [ -vlibs <path_to_library> ] | (optional) Use the Verilog library option specified by path_to_library. This option is not needed if the variable CDS_INTFC is declared in your environment. |
| -noio | (optional) Do not add IBUFs, OBUFS, or EXT records to the design. By default, *x2vprep* inserts I/O on loadless and sourceless signals to prevent them from being deleted prior to functional simulation. |
| -save | (optional) Save the intermediate files generated by *funcnetx*. The intermediate files include the .bxa, .blx, .xg, .xcd, .xff, .xtg, .mrg, .prx, and .prp files. By default, these files are removed once *funcnetx* has completed processing of a design. |

Example: funcnetx is most commonly called with only two parameters, input design name, and type of architecture.

```
funcnetx foo 4000
```

In this example, the design name is “foo”, and the architecture is XC4000.

**Note:** Functional simulation of XC7000 designs is not supported by funcnetx.

## Input Files

funcnetx requires the following file:

| | |
|---|---|
| design.xnf | This is a flattened, gate-level XNF file that represents your entire design. |

## Output Files

The files and directories funcnetx produces are described in the following table.

| File/Directory | Description |
|---|---|
| *<design>.mrg* | is the report generated by *xnfmerge*, which is called by *funcnetx*. |
| *<design>.xff* | is similar to the input XNF file, with the following additions:<br>• Input EXT records are added to source-less signals<br>• Output EXT records are added to loadless signals<br>• Bidirectional EXT records are added to bidirectional I/O signals<br>These additions to the XNF file permit you to simulate a sub-block of your design. |
| *<design>f.stim* | is a template Verilog stimulus file, which contains an instantiation of the top-level Verilog module described in <*design>f.v*. This file:<br>• Sets all inputs to 0 (zero) at initialization<br>• Includes a *$gr_waves* statement that includes all inputs and outputs on the top-level module |
| *<design>f.v* | is the Verilog netlist file, which you use as input to the Verilog-XL simulator. It contains instantiations of Xilinx primitives, whose Verilog models are *in <install_dir>/ verilogxxxx*, where *xxxx* represents either 2000, 3000, 4000 or 7000. These models do not contain timing information. |
| *funcnetx.log* | contains output messages from the *funcnetx* script. |

### Function

funcnetx executes the following commands:

| | |
|---|---|
| xnfmerge | |
| xnfprep | (X-BLOX designs only) |
| xblox | (X-BLOX designs only) |
| x2vprep | |
| xnf2verilog | |

## timenetx Script

timenetx is a C-shell script based on the Cadence timenet script shipped with the DS381 interface. timenetx executes the following programs to create files for Verilog-XL timing simulation, as listed in the following table:

| Program | Reference |
|---|---|
| *lca2xnf* (XC2000, XC3000, XC4000) | For information, see the *Development System Reference Guide, Vol. 3* |
| *vmh2xnf* (XC7000) | For information, see the *XEPLD Reference Guide* |
| *xnfba* (XC2000, XC3000, XC4000) | For information, see the *Development System Reference Guide, Vol. 3* |
| *x2vprep* | For information, see the “x2vprep” section in this document |
| *xnf2verilog* | For information, see the “xnf2verilog” section in this document |

If the environment variable, CDS_INTFC, is defined, the location of the pin file for xnf2verilog is assumed to be:

```
$CDS_INTFC/xcXXXXX.pin
```

and the location of the appropriate Verilog library is assumed to be:

```
$CDS_INTFC/verilogXXXX
```

where XXXX is either 2000, 3000, 4000, or 7000. You may also specify the pin file and library locations explicitly using the -pin and -vlibs options.

## Syntax

```
timenetx      design_name
              {2000 | 3000 | 4000 | 7000}
              [-pin pin_file]
              [-vlibs path_to_library]

              (-o output_file]
              [-x]
              [-cds]
              [-r run_dir]
```

## Options

- *design_name* is a timing-annotated LCA, VMD, or VMH file.
- {2000 | 3000 | 4000 | 7000} is the target architecture family of your design. Timenetx determines the appropriate flow to use based on the target architecture.
- [ *-pin pin_file* ] (optional) specifies the location of the pin file, which must be specified by an explicit path starting from the root directory. The pin file name corresponds to the Xilinx architecture that you are targeting. Valid pin file names are xc2000.pin, xc3000.pin, xc4000.pin, or xc7000.pin for XC2000, XC3000, XC4000, or XC7000 devices, respectively. The pin files contain information on how to properly connect unconnected pins found in the XNF file when generating the Verilog code. The pin files are normally located in the directory, install_dir/data/. If the environment variable CDS_INTFC is defined, this option need not be specified, as the pin file will be assumed to be in the $CDS_INTFC/data directory. To specify a pin file location that is different from this, use this syntax:

  ```
  -pin new_dir/xc_family.pin
  ```

  where new_dir is the directory where the pin file is located, and xc_family.pin must be replaced with xc2000.pin, xc3000.pin, xc4000.pin, or xc7000.pin.

Example:

```
-pin /tools/xact/data/xc7000.pin
```

- [ -*vlibs path_to_library* ] (optional) specifies the location of the Verilog library for XNF2Verilog. If the environment variable CDS_INTFC is defined, this option need not be specified. The default location of the Verilog libraries assumed by timenetx is the following:

  ```
  $CDS_INTFC/verilog_family
  ```

  where verilog_family is either verilog2000, verilog3000, verilog4000, or verilog7000.

  [ -o *output_file* ] (optional) specifies the name to be used for the output .v, .stim, and .sdf files. This output file name must be different from the input cellname. The default is *input_file*t.

  [ -x] (optional) This option instructs timenetx to skip XNFBA. The timenetx script will invoke this option automatically if it detects a Synopsys block in the design, as back annotation of original signal and symbol names is not supported for Synopsys designs. The -x option is ignored for XC7000 designs. (XNFBA is not part of the back annotation flow.)

  [ -cds ](optional) This option instructs timenetx to skip the Xilinx programs (LCA2XNF, XNFBA, VMH2XNF).

  [ -r run_dir](optional) The -r option instructs timenetx to use the specified run directory when generating the design netlist files. The default is the current working directory.

  Example:

  ```
  -r xilinx.run
  ```

  The -r option instructs timenetx to write all design netlist files to the directory names xilinx.run

## Input Files

timenetx requires the following files:

- A timing-annotated file: design.lca, design.vmd, design.vmh, or design.xnf.

## Output Files

timenetx creates the following files:

- *output_file*.sdf contains the delay information to be annotated to the Verilog primitives.
- *output_file*.stim is a template Verilog stimulus file, which contains an instantiation of the top-level Verilog module described in output_file.v. This file sets all inputs to 0 (zero) at initialization and includes a $gr_waves statement that includes all inputs and outputs on the top-level module.
- *output_file*.v is the Verilog netlist file, which you use as input to the Verilog-XL simulator. It contains instantiations of Xilinx primitives, whose Verilog models are in *install_dir*/*verilog_family*, where verilog_family is either verilog2000, verilog3000, verilog4000, or verilog7000. This file does not contain timing information; all timing information is stored in the SDF file.
- *design*t.xbf is similar to *design*t.xnf, with the addition of the original symbol and signal names from your schematic.
- *design*t.xnf is a gate-level XNF representation of your implemented design. It contains timing information, but no partitioning information.
- timenetx.log contains output messages from the timenetx script.
- xnf2verilog.info contains output messages from the xnf2verilog program.
- xnfba.rpt contains the report generated by the XNFBA program.

## Example:

```
timenetx foo.xnf 4000
```

In this case, the design, foo, is an XC4000 design. Timenetx reads the input file is foo.lca, and the output file is foo.v. The XC4000 pin file specified in this example is located in /tools/xact/data, and the library being used is in /tools/xact/verilog4000.

## Function

timenetx executes the following commands.

```
lca2xnf -wg
```

```
vmh2xnf -n
xnfba unrouted_xnf routed_xnf -o routed_xnf
xnf2verilog
```

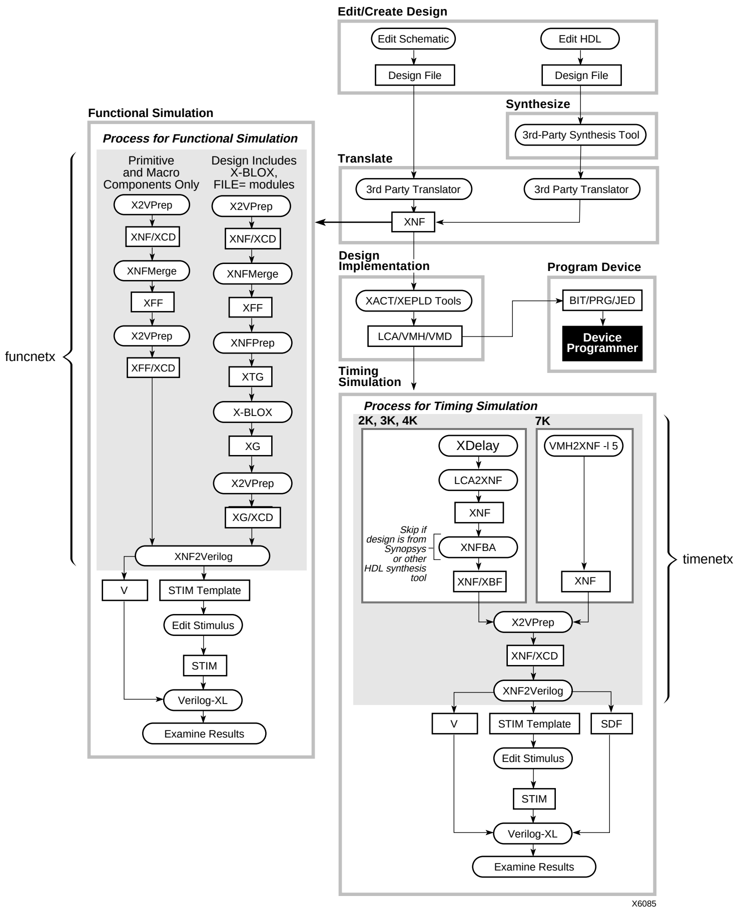
## Detailed Design Flow for Design Verification

Design verification includes both functional and timing simulation. Functional simulation enables you to verify the logic of a design before implementation. Timing simulation enables you to examine the timing delays produced during implementation.

Although the following sections discuss functional and timing simulation design flows in detail, you may use the simulation scripts funcnetx and timenetx to automate the process.

The following figure illustrates how design verification fits into the overall design flow. The main steps are:

1. Edit / Create the Design using schematics or HDL (hardware description language).
2. For designs entered in HDL, the next step is Synthesis of the design.
3. Translate the design to XNF.
4. *Functional Simulation*. This verifies the design's basic functionality without taking delays into account. Note that there are two subflows, one for designs containing X-BLOX and / or FILE= modules, and one for designs containing only primitives and macros.
5. Implement the Design. The implementation step is carried out using the XACT and / or XEPLD tools.
6. *Timing Simulation*. The Xilinx Interface to Verilog-XL allows you to run timing simulation of your design to verify delays and performance.
7. Program the Device. After timing verification is complete, the device can be programmed with the configuration data.



Consult the *Development System Reference Guide* or the *XEPLD Reference Guide* for information on the following additional steps:

1. Performing static-timing analysis.
2. Editing the actual physical implementation of the design.
3. Downloading your design to a Xilinx FPGA or EPLD.
4. Using a PROM to configure a Xilinx FPGA.

## Functional Simulation

This section shows you how to prepare your design to ensure that it simulates correctly and how to use the Verilog-XL simulator to functionally simulate your design.

Xilinx strongly recommends that you functionally simulate your design before you implement it. Functional simulation saves time in the design cycle by identifying logical errors early in the design process.

As the name implies, functional simulation only checks design functionality. To find routing-related timing errors in your design, you can perform timing simulation after you implement the design. Timing simulation is described in the "Timing Simulation" section.

If functional simulation yields satisfactory results, you can proceed to the design implementation step. If not, determine the source of the errors, correct them in your design, and repeat functional simulation until you achieve satisfactory results.

**Note:** Please note the following constraints on functional simulation:

1. Functional simulation of designs done in pre-unified libraries is not supported by this package. You must use the 9402 interface to process pre-unified designs for functional simulation.
2. Pre-route, post synthesis simulation of Synopsys designs is not supported.
3. Functional or unit delay simulation of XC7000 designs is not supported. Refer to the chapter on timing simulation for details on simulating XC7000 designs with routing delays.

### Preparing a Design for Functional Simulation

Before you functionally simulate your design, complete the following steps:

1. If you are using schematics for design entry, ensure that the design does not contain any CLB or IOB primitives. Unlike macros, which are composed of primitives, CLB and IOB primitive symbols do not have underlying simulation primitives and therefore cannot be simulated functionally.

2. If applicable, configure the “hidden” global set / reset pin on flip-flop primitives to the design’s set / reset signal, as described in your third-party design entry user guide.
3. If applicable, configure the “hidden” global 3-state pin on output primitives to the design’s general 3-state signal, as described in your third-party design entry user guide.

## Design Flows for Functional Simulation

There are two basic Verilog-XL functional simulation design flows for Xilinx designs. The flow that you use depends on the elements that comprise your design. Designs with X-BLOX elements follow a different functional simulation flow from designs without X-BLOX elements. This section describes Verilog-XL functional simulation flows for both types of designs.

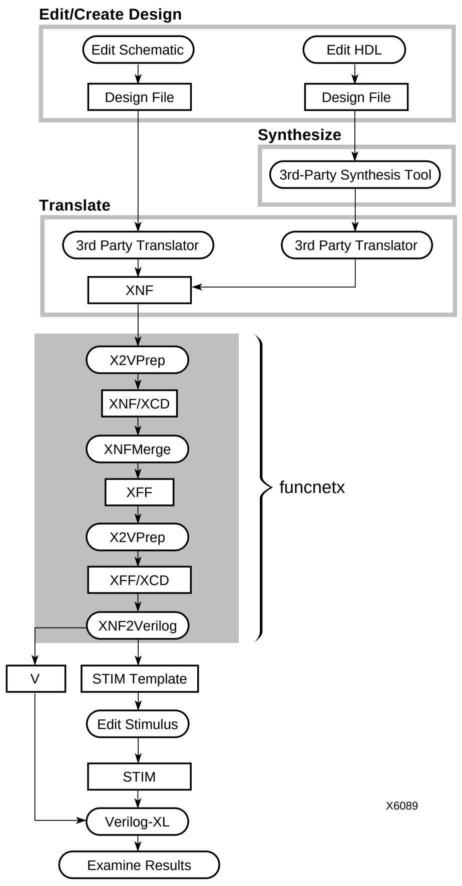
Xilinx recommends that you use the funcnetx script to process your design for functional simulation. The funcnetx script will pick the appropriate flow for you automatically based on the target architecture and types of elements found in your design.

The following sections describe the design flows in detail for users who wish to process their design for simulation manually.

For additional information about using Verilog-XL, see the *Verilog-XL Configuration Guide*, the *Verilog-XL Reference Manua*l, or the Verilog-XL Tutorial.

## Detailed Design Flow for Functional Simulation of Designs with Only Primitive and Macro Elements

The design flow for using the Verilog-XL simulator to functionally simulate designs containing only primitive or macro library components is shown in the following figure.



The procedure is as follows:

1. Run X2VPrep. X2VPrep adds inputs and outputs to your design temporarily if you are processing only a sub-block of your design. Note that the output file, design.xcd, has an XCD extension to prevent X2VPrep from overwriting the original XNF file. Specifying the XNF extension for the input file is optional.

   ```
   x2vprep design[.xnf] designf.xcd
   ```

2. Merge the design to produce a flattened XFF file. Use the following command:

   ```
   xnfmerge design.xcd design[.xff]
   ```

3. Run X2VPrep to convert INV attributes to inverter symbols and to convert the netlist to the proper subset of XNF for XNF2Verilog.

   ```
   x2vprep design.xff design.xcd
   ```

   Again, a .xcd extension is added to the output filename, design.xcd, to prevent X2VPrep from overwriting the original XFF file.

4. Run XNF2Verilog to generate the Verilog source, stimulus, and standard delay format files.

   ```
   xnf2verilog designf.xcd designf -arch \
   architecture -pin path_to_pinfile \
   -vlibs path_to_library_directory
   ```

   Example:

   ```
   xnf2verilog mydesignf.xcd mydesignf -arch 4000 \
   -pin /tools/xact/data/xc4000.pin -vlibs /tools\
   /xact/verilog4000
   ```

   This example uses the XC4000 architecture. The explicit path to the pin file, xc4000.pin, is /tools/xact/data/xc4000.pin. The verilog4000 simulation library is located in /tools/xact/verilog4000. The input design name is mydesignf.xcd, and the output files will be named

   ```
   mydesignf.[ext]
   ```

   For example,

   ```
   mydesignf.v
   mydesignf.stim
   mydesignf.sdf
   ```

   In this example

   - *mydesignf.v* is the Verilog source file.
   - *mydesignf.stim* is the Verilog stimulus file.
   - *mydesignf.sdf* is the standard delay format file containing routing delays.

5. Add the stimulus for your design to the stimulus template (STIM) file (mydesignf.stim, in this example).

6. Invoke the Verilog-XL simulator by entering the following at the command line:

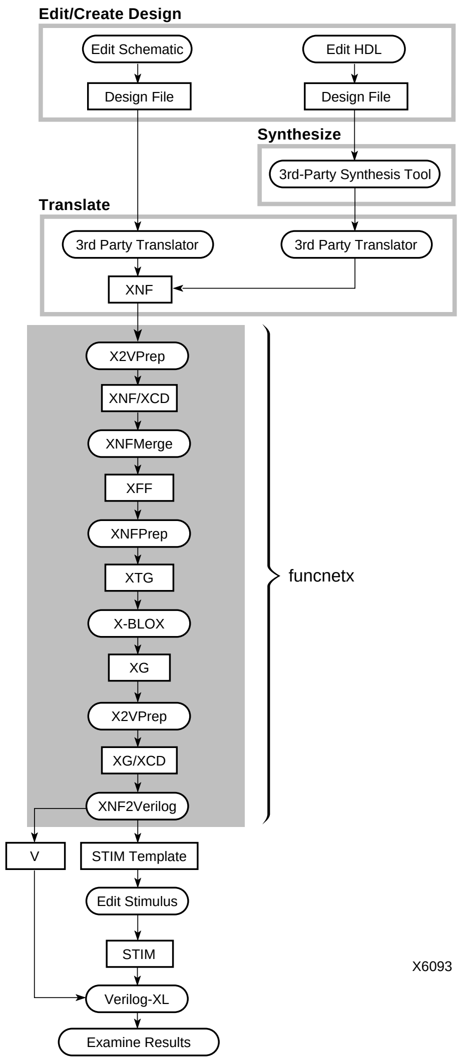
   ```
   verilog designf.v designf.stim [options]
   ```

   This syntax is composed of the following parameters:

   - designf.v is the Verilog-XL netlist created by XNF2Verilog.
   - designf.stim is the stimulus file created by XNF2Verilog.
   - options refers to the Verilog-XL command-line options.

## Detailed Design Flow for Functional Simulation of Designs Containing X-BLOX and/or FILE= modules

The following figure displays the design flow involved in using the Verilog-XL simulator to functionally simulate XC2000 and XC3000 designs containing FILE= modules, and XC3000A and XC4000 designs containing X-BLOX or FILE= modules.



The flow is as follows:

1. Run X2VPrep to add inputs and outputs to your design temporarily if you are processing only a sub-block of your design.

   **`x2vprep design[.xnf] design.xcd`**

Note that the output file, design.xcd, has a .xcd extension to prevent X2VPrep from overwriting the original XNF file. Specifying the .xnf extension for the input file is optional.

2. Merge the design to produce a flattened XFF file. Use the following command:

   ```
   xnfmerge design.xcd design[.xff]
   ```

3. Check the design for errors.

   ```
   xnfprep design.xff design[.xtg]
   ```

4. Run X-BLOX on the design to expand the X-BLOX modules

   ```
   xblox design[.xtg] [design.xg]
   ```

5. Run X2VPrep to convert INV attributes to inverter symbols and convert the netlist to the required subset of XNF for XNF2Verilog.

   ```
   x2vprep design.xg designf.xcd
   ```

   In this case the .xg extension is required for the input file. Also, for the output file, Xilinx recommends that you rename it **designf** and specify a .xcd extension to distinguish the output file from the input .xg file and the output from the first run of X2VPrep.

6. Run XNF2Verilog to generate the Verilog source, stimulus, and standard delay format files.

   ```
   xnf2verilog designf.xcd designf -arch \
   architecture -pin path_to_pinfile -vlibs \
   path_to_library_directory
   ```

   Example:

   ```
   xnf2verilog mydesignf.xcd mydesignf -arch 4000 \
   -pin /tools/xact/data/xc4000.pin -vlibs /tools\
   /xact/verilog4000
   ```

   This example uses the XC4000 architecture. The pin file, xc4000.pin, is located in /tools/xact/data/xc4000.pin. The XC4000 Verilog simulation library is located in /tools/xact/verilog4000. The input design name is mydesignf.xcd, and the output files are named mydesignf.v (Verilog source file), mydesignf.stim (Verilog stimulus file), and mydesignf.sdf (standard delay format file containing routing delays).

7. Add the stimulus for your design to the stimulus template (STIM) file.

8. Invoke the Verilog-XL simulator by entering the following at the command line:

   ```
   verilog designf.v designf.stim [options]
   ```

   This syntax is composed of the following parameters:

   - `designf.v` is the Verilog-XL netlist created by XNF2Verilog.
   - `designf.stim` is the stimulus file created by XNF2Verilog and edited by the user.
   - `options` refers to additional Verilog-XL command-line options you may wish to include.

## Timing Simulation

This section shows you how to use the Verilog-XL simulator to perform timing simulation on your XC2000, XC3000, XC4000, or XC7000 family design.

After implementing your design, you should simulate its timing to ensure that its performance meets your expectations. Timing simulation verifies the timing relationships within the design and determines the critical paths for the design under worst-case conditions. You can also use timing simulation to determine if your design contains setup or hold violations.

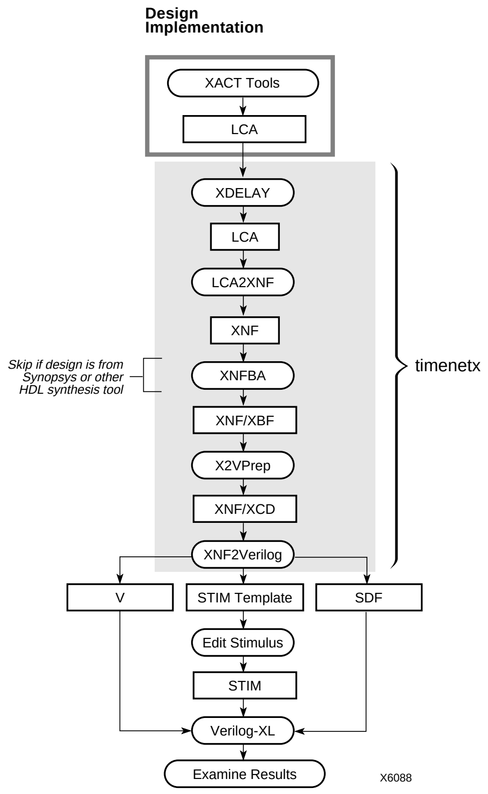
If you determine that the implementation of the design is satisfactory, you can download it to a Xilinx FPGA or program the Xilinx EPLD. If not, you can either re-implement the design, or you can use the XACT Design Editor to alter the actual placement and routing of the design.

It is recommended that you use the timenetx script for processing your design. The timenetx script will select the appropriate flow to use for you based on your target architecture and the types of elements in your design.

The following sections describe the design flows in detail for users who wish to process their designs for simulation manually.

## Detailed Design Flow for Timing Simulation of XC2000, XC3000, and XC4000 FPGA Designs

The flow for processing an XC2000, XC3000, or XC4000 design for post-route timing simulation with the Verilog-XL simulator is illustrated in the following figure. The starting point is a routed LCA with routing delays annotated to it, and the outputs are a Verilog source file (.v), a stimulus template file (.stim), and a delay annotation file (.sdf).



To simulate the timing of your XC2000, XC3000, or XC4000 series design using the Verilog-XL simulator, execute the following steps from your design directory. The overall flow is shown in the figure above.

1. If you did not generate your routed LCA file using XMake 5.x, run XDelay on your design to annotate routing delays to the LCA file.

   ```
   xdelay -dw design[.lca]
   ```

2. Generate the routed XNF file.

   ```
   lca2xnf design[.lca] designt[.xnf]
   ```

   Here, design is the name of the input LCA file, and designt is the name selected for the output XNF file. Xilinx recommends that you append the “t” extension to your routed XNF file to distinguish it from the unrouted XNF file and to avoid overwriting the unrouted XNF file.

3. Run XNFBA to back-annotate net and instance names to the post-route XNF file. (Skip this step if you are simulating a Synopsys design, because you will not have gates to which you can back-annotate.)

   ```
   xnfba design designt[.xnf] -o designt.xbf
   ```

4. Convert INV attributes to INV gates for XNF2Verilog.

   ```
   x2vprep designt.xbf designt.xcd
   ```

   Use the extension .xbf, if you run XNFBA as specified above in step 3. If you skip XNFBA, use the .xnf extension for the input filename.

5. Run XNF2Verilog to generate the Verilog source (.v), stimulus (.stim), and standard delay format (.sdf) files.

   ```
   xnf2verilog designt.xcd outputfilename -s \
   designt.sdf -arch architecture -pin \
   path_to_pinfile -vlibs path_to_library_directory
   ```

   Example:

   ```
   xnf2verilog mydesignt.xcd mydesignt -s mydesignt.sdf\
   -arch 4000 -pin /tools/xact/data/xc4000.pin \
   -vlibs /tools/xact/verilog4000
   ```

6. Add the stimulus for your design to the stimulus template (.stim) file.

7. Invoke the Verilog simulator by entering the following at the command line:

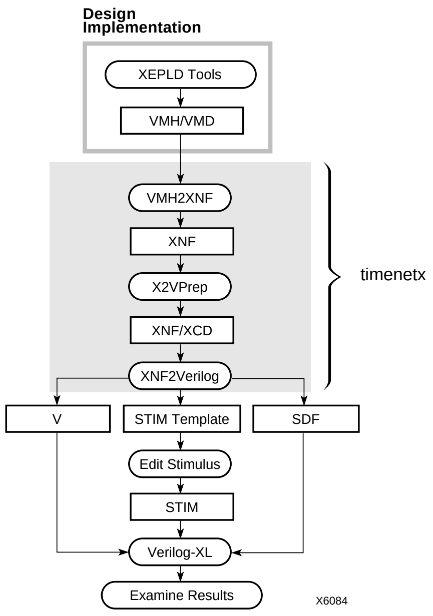
   ```
   verilog designt.v designt.stim [options]
   ```

This syntax is composed of the following parameters:

- designt.v is the Verilog-XL netlist created by XNF2Verilog.
- designt.stim is the stimulus file created by XNF2Verilog.
- [options] refers to the Verilog-XL command-line options.

## Detailed Design Flow for Timing Simulation of XC7000 Designs

The flow for processing an XC7000 design for post-route timing simulation with the Verilog-XL simulator is illustrated in the following figure. The input is a VMH or VMD file, and the outputs are a Verilog source file (.v), a stimulus template file (.stim), and a delay annotation file (.sdf).



The following steps describe the flow you need to follow to do a timing simulation on an XC7000 design using the Verilog-XL simulator.

1. Generate a post-route XNF file from the routed VMH file.

   **vmh2xnf -l 5 -n designt[.vmh] -o design[.xnf]**

2. Convert INV attributes to INV gates for XNF2Verilog.

   ```
   x2vprep designt[.xnf] designt.xcd
   ```

3. Run XNF2Verilog to generate the Verilog source (.v), stimulus (.stim), and standard delay format (.sdf) files.

   ```
   xnf2verilog designt.xcd designt -s designt.sdf \
   -arch architecture -pin path_to_pinfile \
   -vlibs path_to_library_directory
   ```

   Example:

   ```
   xnf2verilog mydesignt.xcd mydesignt \
   -s mydesignt.sdf -arch 7000 \
   -pin /tools/xact/data/xc7000.pin \
   -vlibs /tools/xact/verilog7000
   ```

   In this example, the Verilog interface has been installed in /tools/xact.

4. Add the stimulus for your design to the stimulus template (.stim) file.

5. Invoke the Verilog simulator by entering the following at the command line:

   ```
   verilog designt.v designt.stim [options]
   ```

   This syntax is composed of the following parameters:

   - filename.v is the Verilog-XL netlist created by XNF2Verilog.
   - filename.stim is the stimulus file created by XNF2Verilog.
   - [options] refers to the Verilog-XL command-line options.

**Note:** To initialize an XC7000 simulation, you must first toggle the active low PRLD signal low for 100ns to reset all the flip-flops in the XC7000 device.

## Known Issues

This section describes the known issues and solutions for this Cadence Verilog-XL release.

### Functional Simulation

#### Functional Simulation of Pre-Route, Post-Synthesis XNF Netlists from Synopsys

Platform: All
Architecture: All
Design Step: Functional Simulation
Reference Number: Not Available

Currently, post-synthesis functional simulation of Synopsys designs is not supported. To obtain a post-synthesis gate-level functional simulation netlist, you must place the design using PPR to translate all of the modules into primitives, then follow a timing-simulation netlist flow to generate a netlist that can be simulated. The flow is the following

```
xmake -n design

ppr design route=false placer_effort=1

lca2xnf design designt

x2vprep designt designt.xcd

xnf2verilog -arch family -pin path_to_pinfile -vlibs
path_to_library designt.xcd designt
```

#### XNF2Verilog V9504-1-30B Cannot Resolve All Pin Names in a MemGen-Generated XNF File

Platform: All
Architecture: XC4000
Design Step: Functional Simulation
Reference Number: 22108

XNF2Verilog usually flags the problem with this message:

```
ERROR:

Could not find primitive OR2, pin i2 in
does_pin_have_delay.
```

The Verilog-XL simulator issues this error:

```
Error! Port (i2) not found in module definition
[Verilog-PNFMD].
```

MemGen numbers combinational logic gate input pins starting from I1. This method differs from the way Cadence numbers pins in its Verilog simulation library primitives; these pins start at I0. XNF2Verilog V9502 is unable to resolve the differences in the pin-numbering conventions between MemGen and the Verilog library. Trying to process a design with a MemGen block in it for functional simulation causes the error just given during Verilog simulation.

In timing simulation, this problem appears if the design contains a MemGen block, and you run XNFBA to restore the original pin names.

Since MemGen only generates memory blocks for XC4000 designs, this problem is not encountered in XC2000, XC3000, or XC7000 designs.

As a solution, pre-process the design through XMake, and use PPR to place the design without routing it. Then complete processing with LCA2XNF, X2VPrep, and XNF2Verilog. Since the design is not routed, all delays in the routed XNF file are unit delays. Use this sequence of commands:

```
xmake -n design

ppr design.xtf placer_effort=1 -route=false

lca2xnf design designt

x2vprep design

xnf2verilog
```

Do not run XNFBA, because it back-annotates the original MemGen-style pin names.

## Functional Simulation of Designs Implemented Using the Pre-Unified Libraries Is Not Supported By This Package

Platform: All
Architecture: XC2000, XC3000, XC4000
Design Step: Functional Simulation
Reference Number: Not Available

If your design is entered using pre-Unified libraries, you must use the Cadence 9402 interface to process the design. The application note "Using the Xilinx Interface 4.0 with XACT 5.0" describes the procedure for processing pre-Unified designs in detail.

Contact Cadence technical support at 1-800-223-3622 or Xilinx technical support at 1-800-255-7778 to request a copy.