CORE Generator User Guide version 1.4

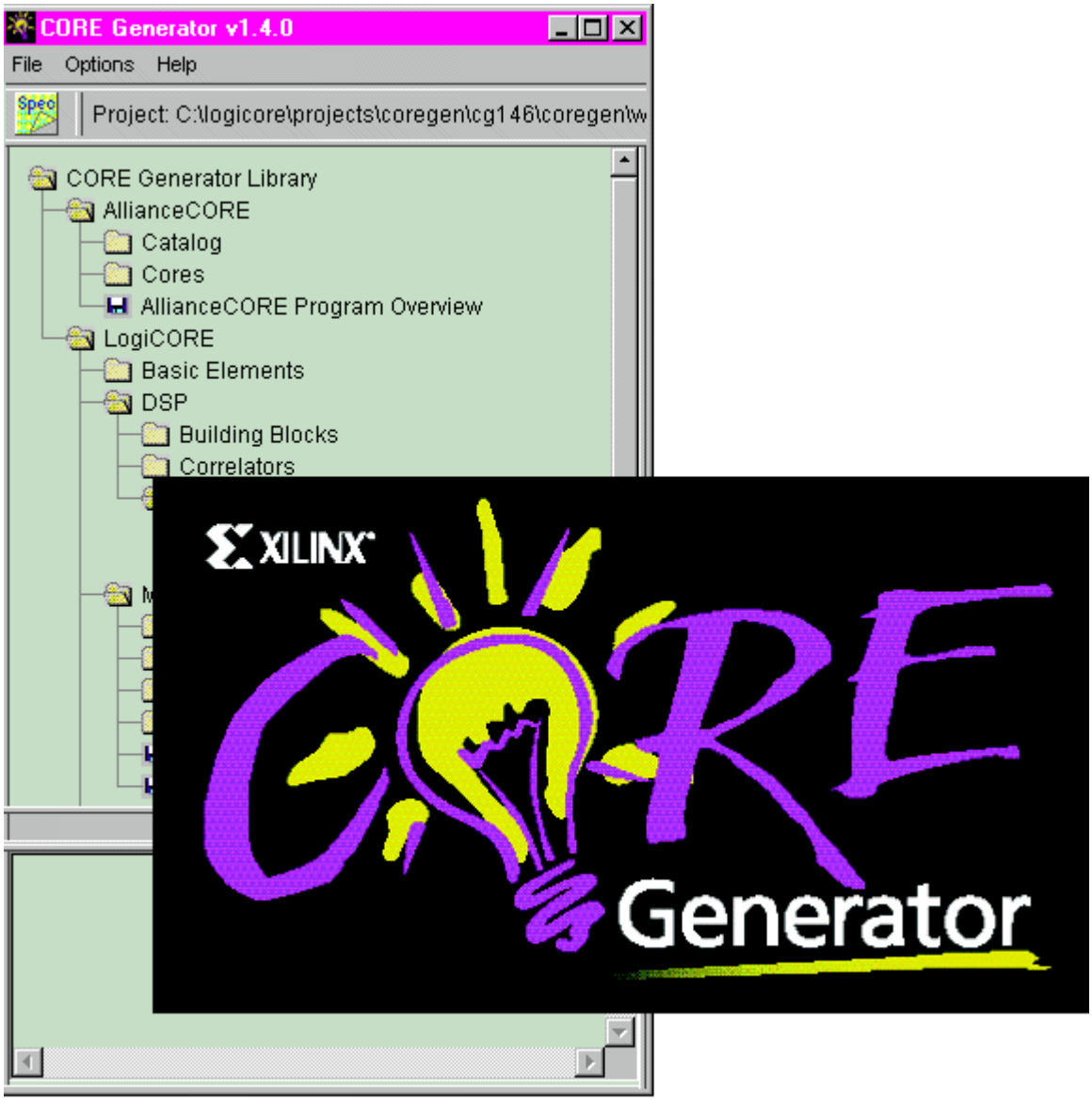
CORE GENERATOR USER GUIDE 1.4

Table of Contents

1 Introduction

1.1 Overview

Welcome to the Xilinx CORE Generator! The Xilinx CORE Generator is an easy to use design tool that delivers parameterizable cores optimized for Xilinx FPGAs.



The Xilinx CORE Generator, presents the designer with a catalog of ready-made functions ranging in complexity from simple arithmetic operations such as adders, accumulators, and multipliers, to system-level building blocks, including filters, transforms, and memories. Selecting the desired function, or module, is simplified by the tree-like presentation of the catalog, where modules of a similar type are grouped together in folders that expand or contract on demand.

Detailed information about any module presented in the browser is instantly available and includes details of how to use the module in your own application, as well as guaranteed area and performance figures. When you see the core you want, select it by single clicking on it. A data sheet on a selected core can be viewed by clicking on the Spec icon on the toolbar (requires Adobe Acrobat to be installed on your machine).

One of the most unique features of the CORE Generator is its ability to tailor a generic functional building block, such as a FIR filter or multiplier, to meet the exact needs of your application, and simultaneously deliver the highest possible levels of performance and area efficiency. These high levels of optimization are guaranteed in your application as a result of both Xilinx's 'Core'-friendly FPGA architectures, and the intrinsic layout (or floorplan) that is imposed on the module. Guaranteed access to the highest possible levels of performance and area efficiency, applied to a diverse library of complex building blocks, makes the design of systems-on-a-chip a simpler and faster process than ever before.

In addition to FPGA implementation details, the CORE Generator delivers behavioral simulation models, schematic symbols and HDL instantiation templates for the modules you choose, allowing the CORE Generator to fit easily into your preferred design environment. The CORE Generator is compatible with HDL and schematic capture design methodologies.

When running the CORE Generator software for the first time be sure to set up your system options (e.g. working directory, etc.) from the **Options -> System Options…** pull down menu and output format options (e.g. generation of VHDL or Verilog, etc.) from the **Options -> Output format…** pull down menu.

You can modify a Module's parameters by double clicking on a core in the hierarchy list. This brings up a parameterization screen which allows you

to define the parameters for each core. Select the parameters appropriate to the function that you need to create.

After defining all of your parameters, you can generate a Core by simply clicking on the **Generate** button. The output is a highly optimized CORE for the targeted FPGA device comprised of the following files:

- A tailored Xilinx netlist with complete relative placement information to guarantee performance
- VHDL or Verilog instantiation code
- A VHDL behavioral model
- A symbol for schematic capture tools

1.2 How to Obtain New Cores and Updates

New cores can be downloaded from the Xilinx web site and easily added to the CORE Generator.

Bookmark:

http://www.xilinx.com/products/logicore/coregen

and keep in touch regularly for updates.

We encourage you to check the CORE Generator WEB page before starting a new design to verify that you have the latest version of the core and core data sheet, as well as to look for any new cores that may be useful in your design.

You must register for CoreLINX before downloading any of the Cores and Updates on the CORE Generator WEB page.

2 System Requirements

2.1 All Platforms

Adobe Acrobat ver 3.0.0 or later (Adobe Acrobat 3.01 is included on CORE Generator 1.4 CD)

2.2 Win95/NT

- PC with Win95 or NT4.0
- 486 or better
- 32MB RAM (Large cores requiring device sizes range from XC4036 through XC4062 will require 64MB -128MB RAM. Cores targeted to larger devices will need up to 128M)

Table 2-1 Memory Requirements for PCs

Xilinx Device	RAM	Swap Space
XC4003E/L through XC4008E/L XC4005XL through XC4008XL XC4000XV XC9500/F (small devices only)	32 MB	32 MB - 64 MB
XC4010E/L through XC4025E/L XC4028EX through XC4036EX XC4010XL through XC4028XL XC4000XV XC9500/F (medium devices only)	64 MB	64 MB–128 MB
XC4036XL through XC4062XL XC4000XV XC9500/F (large devices)	128 MB	128 MB–256 MB

2.3 Solaris

- Solaris 2.5 or 2.6
- Ultra Sparc (or equivalent)
- 64 MB RAM (Cores targeted to larger Xilinx devices may need up to 128M)

Table 2-2 Memory Requirements

Xilinx Device	RAM	Swap Space
XC4000E/L XC4028EX through XC4036EX XC4005XL through XC4028XL XC4000XV XC9500/F (small devices)	64 MB	64 MB–128 MB
XC4036XL through XC4062XL XC4000XV XC9500/F (large devices)	128 MB	128 MB–256 MB

3 Installation Instructions

3.1 Xilinx CORE Generator Requirements for Workstations

The Xilinx CORE Generator Software supports the following workstation architectures and operating systems:

Solaris 2.5 and 2.6

Table 3-1 Memory Requirements

Xilinx Device	RAM	Swap Space
XC4000E/L XC4028EX throughXC4036EX XC4005XL through XC4028XL XC4000XV XC9500/F (small devices)	64 MB	64 MB–128 MB
XC4036XL through XC4062XL XC4000XV XC9500/F (large devices)	128 MB	128 MB–256 MB

Note: The values given in the above table are for typical designs, and include the loading of the operating system. Additional memory may be required for certain *boundary-case* designs, as well as for concurrent operation of other application. Xilinx recommends that 4000EX designs be compiled using an Ultra Sparc, or equivalent machine type.
64MB of RAM as well as 64MB of swap space is required to compile 4000EX designs, but Xilinx recommends that 128MB of RAM, plus corresponding swap space, be used.

3.2 Xilinx CORE Generator Installation on Workstations

Installation of the Xilinx CORE Generator software is completed in two steps.

1. Mount the CDROM labeled Xilinx CORE Generator.
2. Run install located in the CDROM root directory.

For detailed information on how to mount and unmount a CDROM, and how to run the various installation programs, see the *Installation* section of the Release Document.

3.3 Xilinx CORE Generator Requirements for PCs

The Xilinx CORE Generator Software supports the following PC operating systems:

Windows 95 and Windows NT 4.0.

Note: The values given in the above table are for typical designs, and include the loading of the operating system. Additional memory may be required for certain *boundary-case* designs, as well as for concurrent operation of other applications (for example, MS Word or Excel).

Table 3-2 Memory Requirements for PCs

Xilinx Device	RAM	Swap Space
XC4003E/L through XC4008E/L XC4005XL through XC4008XL XC4000XV XC9500/F (small devices only)	32 MB	32 MB - 64 MB
XC4010E/L through XC4025E/L XC4028EX through XC4036EX XC4010XL through XC4028XL XC4000XV XC9500/F (medium devices only)	64 MB	64 MB–128 MB
XC4036XL through XC4062XL XC4000XV XC9500/F (large devices)	128 MB	128 MB–256 MB

3.4 Xilinx CORE Generator Installation on PC

The following table applies to Windows 95 and NT 4.0 installations.

Installation of the Xilinx CORE Generator software is completed in two steps.

1. Insert the CDROM labeled ***Xilinx CORE Generator*** in the CDROM drive.

2. Run **setup.exe** located in the CDROM root directory.

3.5 How to Obtain New Cores and Updates

New cores can be downloaded from the Xilinx web site and easily added to the CORE Generator.

Bookmark:

http://www.xilinx.com/products/logicore/coregen

and keep in touch regularly for updates.

We encourage you to check the CORE Generator WEB page before starting a new design to validate that you have the latest version of the core and core data sheet as well as to look for any new cores that may be useful in your design.

You must register for CoreLINX before downloading any of the Cores and Updates on the CORE Generator WEB page.

4 Project Management

The CORE Generator determines its operational settings from the **coregen.ini** file. This file is created during the installation of the CORE Generator. CORE Generator options may be changed in two ways. The first method is to modify the parameters from the **Options->System_Options** menu from the top of the module browser window. The second method is to modify the parameters in the **coregen.ini** file. Only the modifications to the **coregen.ini** file are permanent. When the CORE Generator options are modified through the options menu the changes are for that session only.

All the files created by the CORE Generator during normal use are stored in the CORE Generator project directory PCS: During installation, the CORE Generator project is created for you in the following directory:

<CORE_Generator_Install_Path>/coregen/wkg

Until the CORE Generator project directory is changed, all files created by the CORE Generator are deposited in this directory. The project directory is displayed at the top of the module browser window at all times to remind you of the current project setting.

4.1 Set up the CORE Generator .INI File (coregen.ini)

A **coregen.ini** file is created during the install of the CORE Generator and can be found in the following directory:

<CORE_Generator_Install_Path>/coregen/wkg

Most of the parameters set in this file are determined during installation, and usually do not need to be changed. However, there are a few parameters that may be altered, or added, to this file.

For example, if you wish to change the CORE Generator's project directory to a location other than the default, then you may set the ProjectPath parameter to point to the new location.

Example: `SET ProjectPath = C:\projects\juke\filter\v1_0`

The result of changing the CORE Generator **coregen.ini** in this way is that each time the CORE Generator is started these settings will be used. If a particular variable or product is not detected during the install process, or you move or install that product later, you will need to modify the **coregen.ini** file. The following parameters may bespecified in the **coregen.ini** file:

ProjectPath - This setting defines the project working directory. All output files produced by the CORE Generator are placed here. When changing the Project Path parameter, all new CORE Generator output files are deposited in this new CORE Generator project directory. The default Project Path that is setduring installation of the CORE Generator is **<CORE_Generator_Install_Path>/coregen/wkg**. Look here for any output files created by the CORE Generator.

Example: `SET ProjectPath = C:\projects\juke\filter\v1_0\wkg`

Options: Path to working directory.

SelectedProducts - This setting defines the type of output files to be created each time a module is built.

Example: `SET SelectedProducts = XNF FoundationSym`

Options: Multiple output formats should be specified on the same line, separated by a blank space. The SelectedProducts parameter options are case sensitive.

- **XNF:** XNF Implementation Netlist. This is the gate level netlist that is used to implement the logic of the particular module that the CORE Generator has created. The HDL templates or schematic symbols created by the CORE Generator point to this netlist.
- **ViewSym:** Viewlogic Schematic Symbol. When this option is specified the CORE Generator will create a Viewlogic schematic symbol that can be used in a Viewlogic schematic to instantiate the module netlist.
- **FoundationSym:** Foundation Schematic Symbol. When this option is specified the CORE Generator will create a Foundation schematic symbol that can be used in a Foundation schematic to to instantiate the module netlist.
- **VHDLSym:** VHDL Instantiation Template. When this option is specified

CORE Generator will create a VHDL instantiation template that can be used to instantiate the module netlist in your VHDL design.

- **VHDLSim:** VHDL Behavioral Simulation Model. When specified, this option will create a VHDL behavioral simulation model, which can be used to verify the functional operation of the module netlist.
- **VerilogSym:** Verilog Instantiation Template. When specified this option will create a Verilog instantiation template that can be used to instantiate the module netlist in your Verilog design.

ViewLogicLibraryAlias - This setting defines the name of the ViewLogic library alias. This alias should be the same alias that is set in the view-draw.ini file.

Example: `SET ViewlogicLibraryAlias = primary`

Options: Any valid alphanumeric 8 character name.

FoundationPath - This setting defines the path location of the Foundation CAE tools. It must be specified for Foundation symbols and library files to be created.

Example: `SET FoundationPath = C:\fndtn\active`

Options: Single path to the installation of the Foundation CAE tools.

AcrobatPath - This setting defines the path location of the Acrobat tools. It must be specified for the proper launching of the data sheets and help documentation.

Example: `SET AcrobatPath = C:\Acrobat3\Reader\`

Options: Single path to the installation of the Acrobat tools.

AcrobatName - This setting defines the name of the Acrobat executable.

Example: `SET AcrobatName = AcroRd32.exe`

Options: Single name of an Acrobat executable.

The values of the SET options, XACTPath, ProSeriesPath, and ViewLogicPath below are not used in version 1.4.x of the CORE Generator, however, they must be present, and should not be removed.

```
SET XACTPath = c:\xact
SET ProSeriesPath = c:\proser
SET ViewlogicPath = c:\wvoffice
```

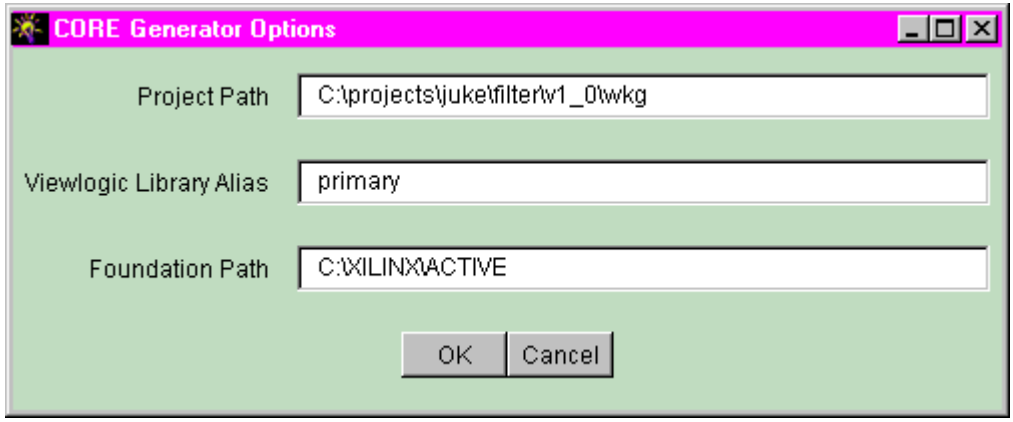
Advanced: Multiple COREGEN.INI Files

When the CORE Generator is launched, it looks for a **coregen.ini** file in the current working directory. For Windows 95 and NT applications the current working directory is defined in the short-cut that is used to launch the CORE Generator. You may therefore edit the short-cut that you use to launch the CORE Generator (usually the CORE Generator entry on the Windows Start-Bar) to set its 'Working Directory' field to a project directory that contains its own **coregen.ini** file. The settings in this **coregen.ini** file will then define the project directory (and any other settings) that the CORE Generator assumes on start-up.

Alternatively, you may create several short-cuts, each with its working directory pointing to a different project directory, in which a **coregen.ini** file specifies the CORE Generator parameters specific to each project.

4.2 Set the CORE Generator System Options

From within the CORE Generator, session options are set by selecting the **Options-> System_Options** menu from the top of the module browser window. Be aware that the options set through this option menu are only applicable to the current CORE Generator session. To make these options permanent, you will want to modify your **coregen.ini**. In this menu selection you will find the following options in the Sample GUI shown below:



Project Path - This setting defines the project working directory. All output file produced by the CORE Generator will be placed here. If you change the Project Path parameter then all new CORE Generator output files will be deposited in the new CORE Generator project directory. The default

Project Path that is setduring installation of the CORE Generator is **<CORE_Generator_Install_Path>/coregen/wkg**. Look here for any output files created by the CORE Generator.

Example: `Project path setting = C:\projects\juke\filter\v1_0\wkg`

Options: Path to working directory.

Viewlogic Library Alias - This setting defines the name of the ViewLogic library alias. This alias should be the same alias that is set in the viewdraw.ini file.

Example: `Alias value used in example GUI = primary`

Options: Any valid alpha numeric 8 character name, default value is "primary".

Foundation Path - This setting defines the path location of the Foundation CAE tools. It must be specified for Foundation symbols and library files to be created.

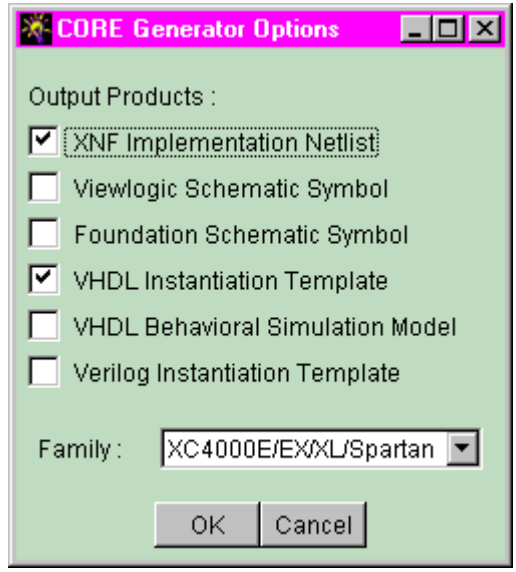
Example: `Foundation Path` setting: `C:\xilinx\active`

Options: Single path to the installation of the Foundation CAE tools.

You need to remember that changing the CORE Generator project parameters in this fashion is only a temporary change since the next time the CORE Generator is launched this new setting will be lost. To make a more permanent change to the CORE Generator project parameters, see the section entitled *Set The CORE Generator .INI File*.

4.3 Set The CORE Generator Output Options

The CORE Generator can create several different types of output. The desired outputs for a particular project may be selected from the **Options->Output Products** window.



Any combination of output formats may be requested at any time. The default selections are XNF Implementation Netlist and VHDL Instantiation Template. The following is a description of each file format:

XNF Implementation Netlist - This is the gate level netlist that will be used to implement the logic of the particular module that the CORE Generator has created. The HDL templates or schematic symbols will point to this netlist. The XNF output option should always be selected.

Output: `<ModuleName>.xnf`

Viewlogic Schematic Symbol - When selected, this option will create a Viewlogic schematic symbol and a simulation wir file that can be used in your ViewLogic schematic capture tools to instantiate the module netlist.

Output: `wir\<ModuleName>.1`
`sym\<ModuleName>.1`

Foundation Schematic Symbol - When selected, this option will create a Foundation schematic symbol and simulation file that can be used in your Foundation schematic capture tools to instantiate the module netlist.

Output: `<ModuleName>.alr`
`lib\project_name.sym`

VHDL Instantiation Template - When selected this option will create a VHDL instantiation template that can be used to instantiate the module netlist in your HDL design capture tool.

Output: `<ModuleName>.vhi`

VHDL Behavioral Simulation Model - When selected this option will create a VHDL simulation model, which can be used to verify the functional simulation of the module netlist. This file is not intended to be synthesized. It is only provided for behavioral simulation. Any attempts to synthesize this netlist will yield sub-optimal results.

Output: `<ModuleName>.vhd`

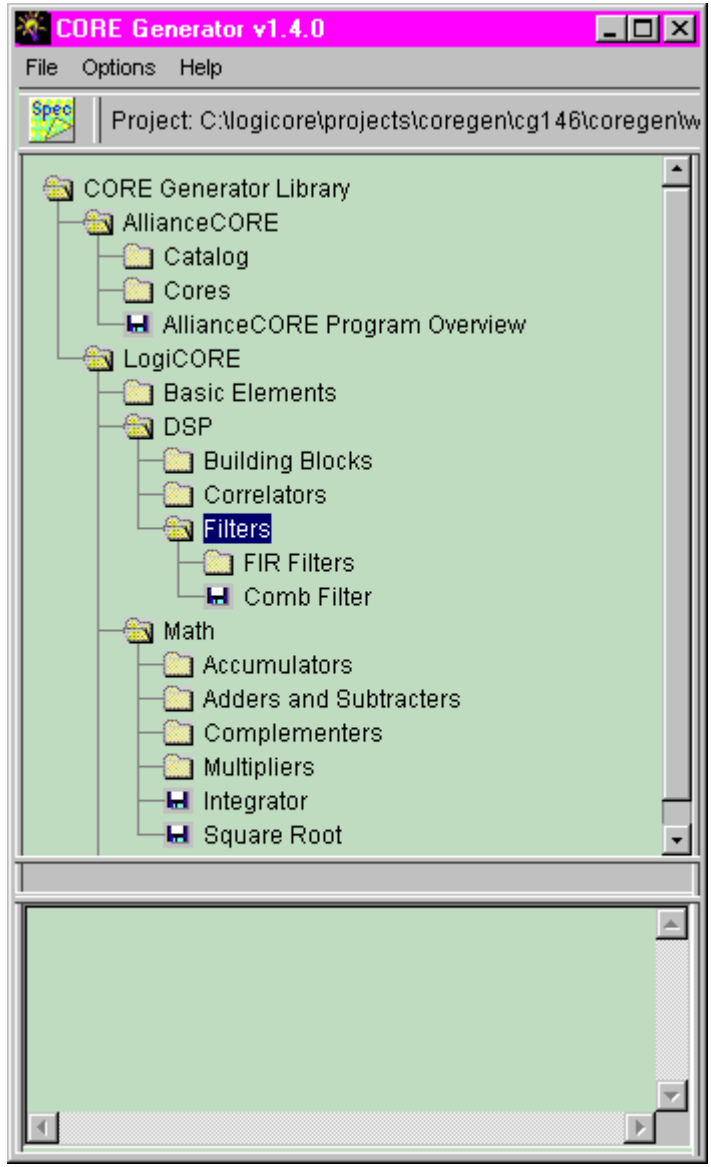
Verilog Instantiation Template - When selected this option will create a Verilog instantiation template that can be used in your HDL design capture tool to instantiate the module netlist.

Output: `<ModuleName>.vei`

The Family drop-down box allows you to restrict the CORE Generator's module browser to show only those modules that may be targeted to the selected family of devices. At this time the supported families of devices are the XC4000E/EX/XL/XV and Spartan.

For detailed information about how to use these files with a variety of CAE design flows, see the **Design Flows** section.

5 Using the CORE Generator



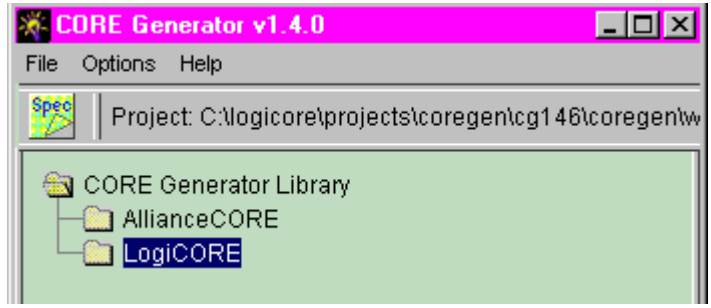
5.1 Module Browser Tree

The most common view of the CORE Generator is the **module browser** window. This window allows you to browse the many modules that are available from the CORE Generator installation. Modules that fall into par-

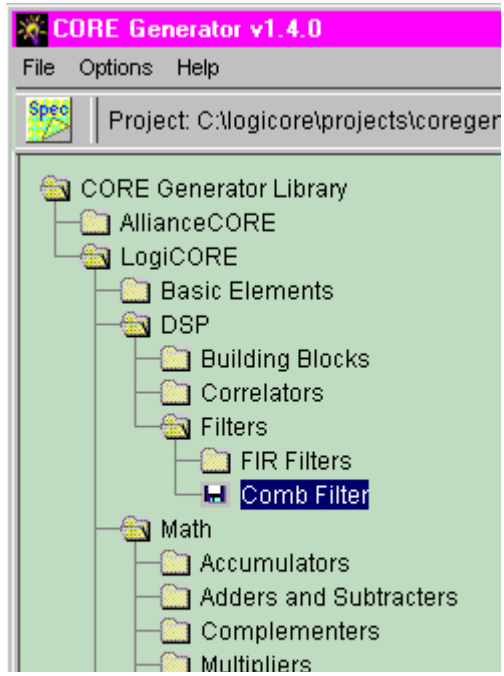
ticular application categories are grouped into folders to assist you in locating the module appropriate for your needs.

To expand a folder, double click on the folder icon to the left of the folder name or folder name. The folder will expand to reveal more folders, or modules.

Note: The folder may be ‘tidied-up’ by double clicking once more on the now open folder icon - causing it to contract.

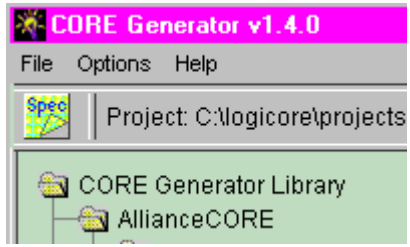


Finally, when the desired module has been located, it can be selected by single clicking on its icon.



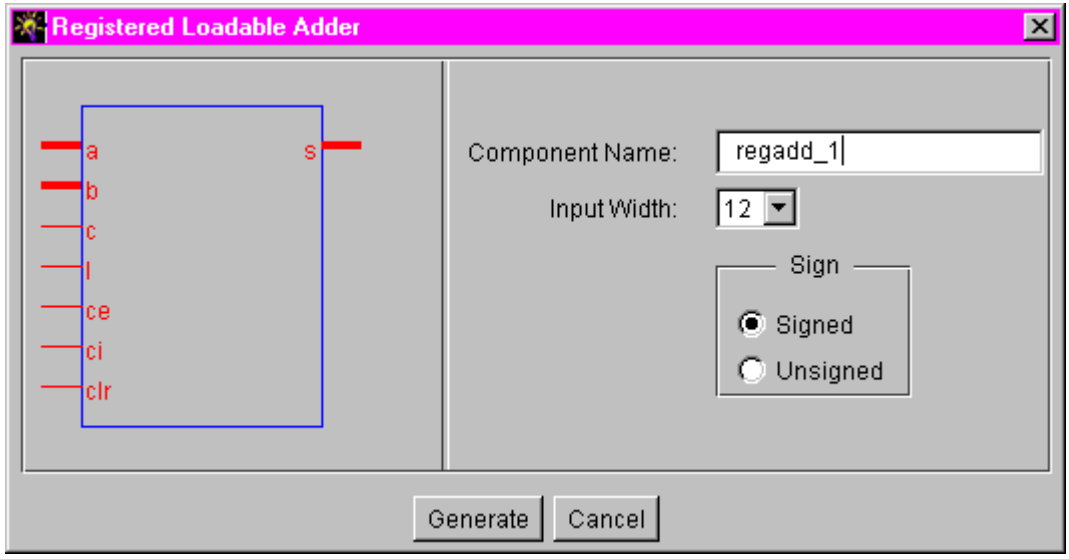
5.2 Getting Module Data Sheets

A data sheet for the selected module can be requested at any time by first selecting the module in the module browser, and then clicking on the SPEC button on the CORE Generator toolbar. This action will launch the Acrobat Reader application and display the module's data sheet.



5.3 Parameterizing a Module

Most modules have a parameterization window. The parameterization window shown below was launched by navigating through the CORE Generator's module browser to the 'Registered Loadable Adder' entry. Double-clicking on a module's icon or descriptive text will reveal the parameterization window for that module. While the parameterization windows will be unique for each module, there are some characteristics that are common to all.



For example, the **Component Name field** allows you to assign a name to a module that you create. Files that the CORE Generator creates for a particular module will have a root filename that matches the Component Name. You should note that **Component Names** have the following restrictions:

- Up to 8 characters
- No extensions
- Must begin with a alpha character: a-z (No Capital letters)
- May include (after the first character): 0-9, _

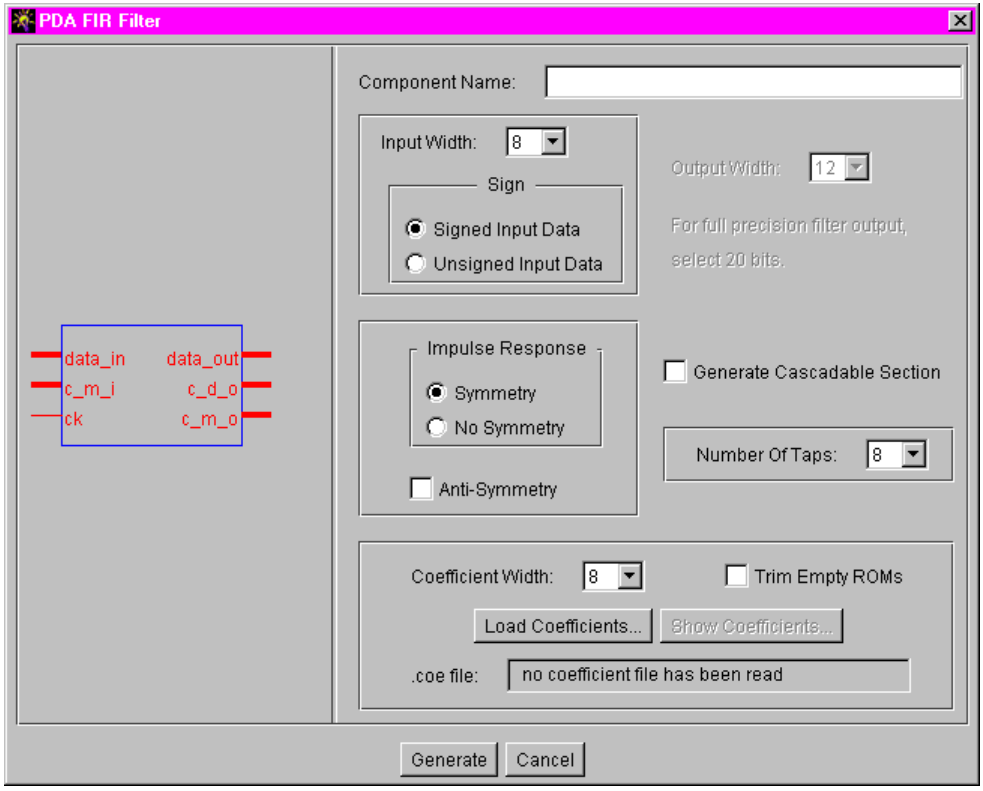
All parameterization windows respond to fields containing illegal or invalid data in the same fashion. The affected field will be highlighted in red until the problem is corrected. If the reason why a field is highlighted is not obvious, or if the explanation in the log window is not clear, a more detailed explanation can usually be obtained by pressing the **Generate** button. Another feature common to all parameterization windows is the **Generate** and **Cancel** buttons. Assuming there are no problems with any of the parameters that have been specified, pressing **Generate** will cause the CORE Generator to create files of the requested types. Pressing **Cancel** will return you to the module browser window without generating any files.

For information about a specific module's parameterization window, such as upper and lower limits for certain fields (or to obtain area/performance figures for specific parameter combinations), see the module's data sheet. General guide lines are discussed in the COE Files section below.

5.4 COE Files

Some modules can require a significant amount of information to completely specify their behavior. Modules of this type will have a button on their parameterization windows which you can use to load their parameterization information from a file. One such module is the FIR filter, whose parameterization window is shown below.

Additional information about a particular Module's COE file can be found in that Module's datasheet. For examples of PDA FIR, RAM, and ROM COE files, please look in the `/coregen/wkg` directory.



.COE files should be ASCII text files and have an extension of .COE. The format for the .COE file is illustrated below:

```
Keyword = Value ; Optional Comment
Keyword = Value ; Optional Comment
…
…
CoefData = Data_Value, Data_Value, …;
```

Note: CoefData or MemData keywords must always be the last Keywords of the file at the end if the file as any further keywords will be ignored. Any text after a semicolon is treated as a comment and will be ignored.

If you are working on a PC the .COE files should be in a DOS format. On a UNIX Workstation UNIX format should be used.

An example .COE file that might be used to parameterize a FIR filter is shown below.

```
********** EXAMPLE: PDA FIR **********

component_name=fltr16;
Number_of_taps=16;
Input_Width = 8;
Signed_Input_Data = true;
Output_Width = 15;
Coef_Width = 8;
Symmetry = true;
Radix = 10;
coefdata=1,-3,7,9,78,80,127,-128;
```

An example .COE file that might be used to parameterize RAM is shown below.

```
********** EXAMPLE: RAM **********

component_name=ram16x12;
Data_Width = 12;
Address_Width = 4;
Depth = 16;
Radix = 16;
memdata=346,EDA,0D6,F91,079,FC8,053,FE2,03C,FF2,02D,FFB,022,002,
```

```
01A,005;
```

An example .COE file that might be used to parameterize a ROM is shown below.

```
********** EXAMPLE: ROM **********

component_name=rom32x8;
Data_Width = 8;
Address_Width = 5;
Depth = 32;
Radix = 10;
memdata=127,127,127,127,127,126,126,126,125,125,125,4,3,2,0,-1,-2,-4,-
5,-6,-8,-9,-11,-12,-13,-38,-39,-41,-42,-44,-45,-128;
```

6.1 Viewlogic Schematic Flow

In order for the CORE Generator to successfully generate all the necessary files for this flow, it is required that the Viewlogic tool and the Xilinx M1 software be set up properly on the same machine the CORE Generator is running on. If one of these tools is missing or not set up properly, errors will be breported in the vllink.log file located in the project directory. Please refer to the **Alliance Quick Start User Guide, Appendix E** for Viewlogic setup instructions.

1. Create a directory for your ViewLogic project.

   Eg: `c:\viewlog\project`

2. Set up the project libraries.

   On Workstations, these libraries must be defined in the viewdraw.ini file located in the project's working directory.
   On PCs, these libraries must be defined in the Viewlogic project file (.VPJ), located in the project's working directory.

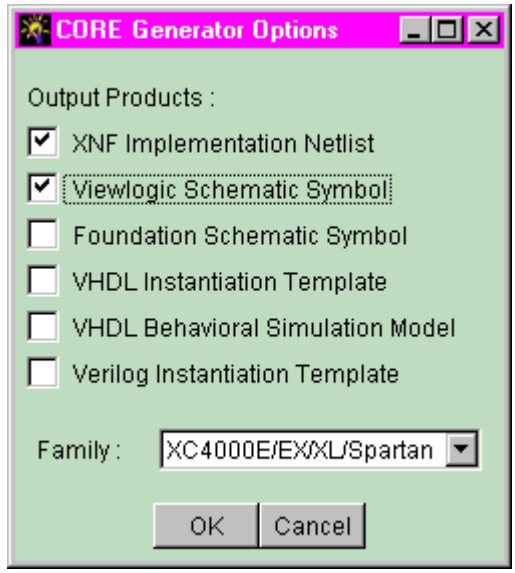
   The following is the library search order needed to create an XC4000XL design:

   ```
   dir [p] c:\viewlog\project (primary)
   dir [rm] %XILINX%\viewlog\data\xc4000xl (xc4000xl)
   dir [r] %XILINX%\viewlog\data\logiblox (logiblox)
   dir [rm] %XILINX%\viewlog\data\simprims (simprims)
   dir [rm] %XILINX%\viewlog\data\builtin (builtin)
   dir [rm] %XILINX%\viewlog\data\xbuiltin (xbuiltin)
   ```

   Note: The (primary) alias is very important since the CORE Generator uses it to determine to which directory the symbol and simulation files will be copied. This alias should match the one specified in the Core-Generator System Options in the Viewlogic Library Alias field.

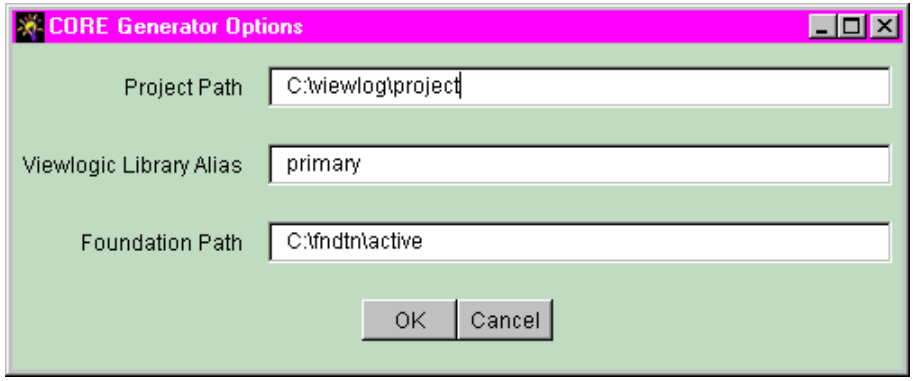
3. Set the Output Format.

From the CORE Generator Options menu, select Output format and check the following options:

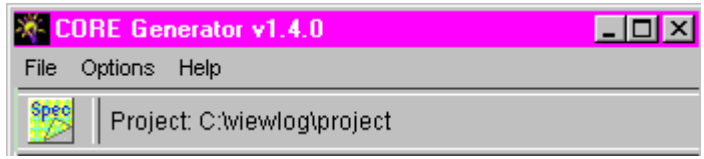


4. Set the Project Path and the Viewlogic Library Alias

   From the CORE Generator Options menu, select **System Options** and set the **Project Path**to point to your Viewlogic projectdirectory (`c:\viewlog\project`). Also make sure that the ViewLogic Library Alias for your project matches the one defined in the `viewdraw.ini` file. The default ViewLogic Library Alias is *primary,* but any name of 8 characters can be used.



5. Select the module you want to generate by navigating to the desired module browser and clicking on it. You can click on the SPEC button on the CORE Generator toolbar to review the modules datasheet.



Double-click on the selected module to call up its parameterization window.
When you have entered all the parameterization details required by the module, press the **Generate** button.

6. Output Files

A Viewlogic Symbol, a simulation file and a Netlist File (.XNF) are generated.

- The symbol of typeComposite is created and placed in the SYM subdirectory under the Viewlogic project directory.
- The simulation file is created from the XNF file and placed in the WIR subdirectory under the Viewlogic project directory.
- The Netlist (.XNF) used for implementation, is placed directly in the ViewLogic project directory.

Note: The WIR file is used by Viewlogic to perform Functional simulation and should never be deleted. In order to generate this file, the CORE Generator needs to access some Xactstep M1 executables and may fail if this tool is not set up properly.

Check the vllink.log file located in the Project Directory if an error occurs during the generation of these files.

7. Load the Symbol in the Schematic Editor

Open the ViewLogic schematic tool, load your top level schematic (or create a new one) and add the new symbol for the module you have just created. From this point on, the flow for processing this design is the same as if you were using macros from the Unified Library. Please refer to the **Viewlogic Tutorial Guide** for further information.

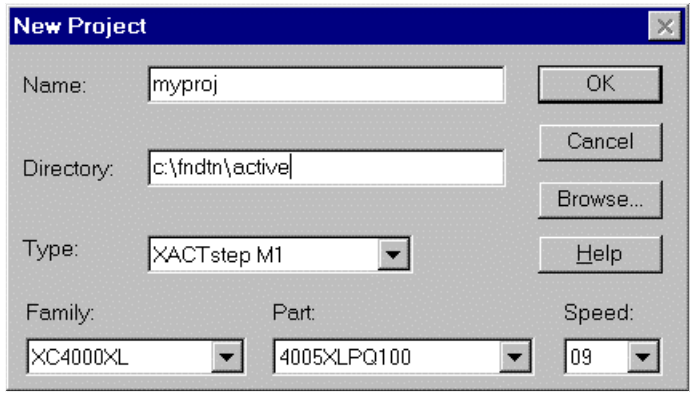
Note: When executing the Viewlogic “Check” program, the following error messages will be displayed for every COREGen module in the design. It can be safely ignored:

Example:

```
ERROR: Could not load schematic sheet: corename.1
```

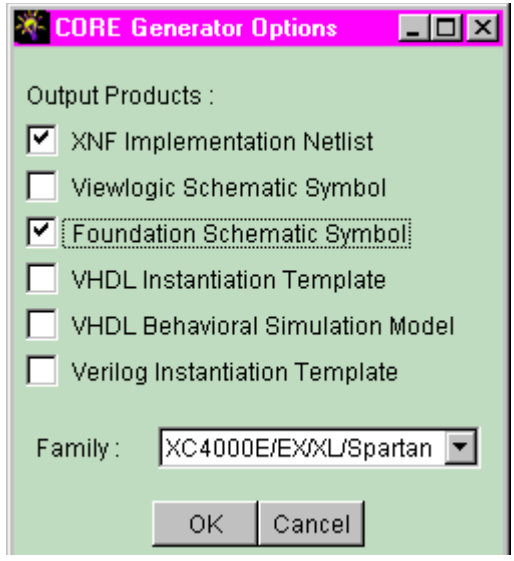
6.2 Foundation Schematic Flow

1. Set a Foundation Project
   Create a new project or Select an existing Project from the Foundation Project Manager.



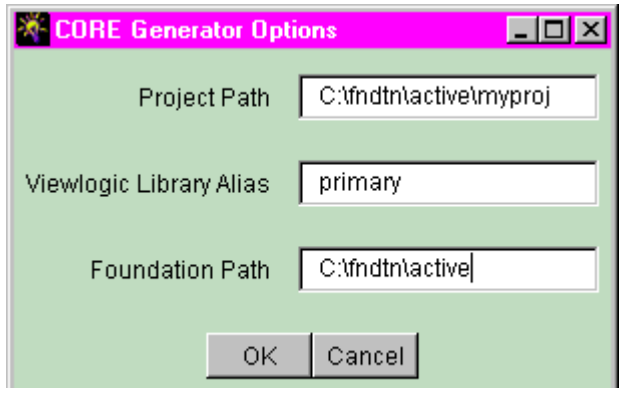
   The files generated by the CORE Generator will automatically be copied into the selected project directory.

2. Set the CORE Generator Output Format
   From the CORE Generator **Options** menu, select **Output Format**, and check the following options:



3. Set the System Options

   From the CORE Generator Options menu, select **System Options**, and set the **Project Path** to point to your Foundation project directory. If the Project Path points to a directory other than the one set in the Foundation Project Manager, the CORE Generator will copy the files in both locations, except for the symbol, which will only reside in the Foundation Project directory.



   While you are in this menu, make sure that the Foundation Path is set correctly. This path should point to the directory where Foundation has been installed.

4. Select the module you want to generate by navigating to the desired module and clicking on it.
   You may click on the SPEC button on the CORE Generator toolbar to review the module's datasheet, or double-click on the selected module to call up its parameterization window. When you have entered all the parameterization details required by the module, press the **Generate** button.

5. Output Files

   A Foundation symbol, a Xilinx Netlist File (.XNF) and a simulation file (.ALR) are created. The symbol is automatically copied in the Foundation Project directory and can be added to the top level schematic from the Symbol menu.

6. Load the Symbol in the Schematic Editor

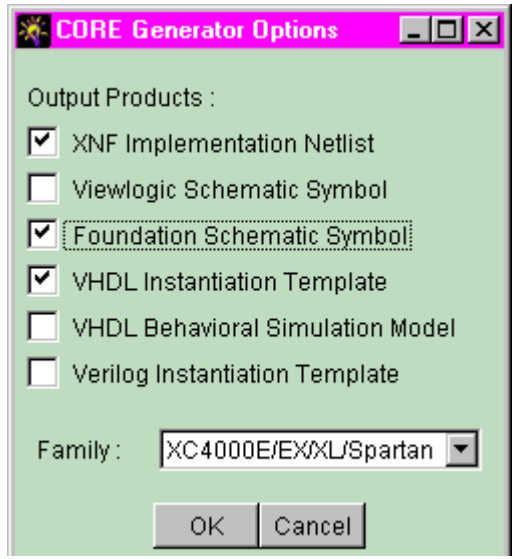
Open the Foundation schematic editor, load your top level schematic (or create a new one) and add the new symbol for the module you have just created. The new symbols will be found in the library list along with the Unified Library symbols.

Note: It may be necessary to select **File->Update Libraries** from the Schematic Editor to be able to view the newly created cores in the library list.

The Symbol can now be added into the Top Level schematic like any other symbol. From this point on the simulation and compilation flow is the same as the Unified Library components.

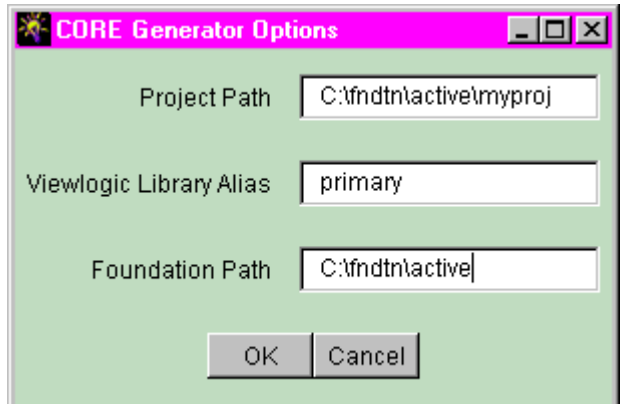
6.3 Foundation Express Flow

1. Create a New Project, or Select an existing Project from Foundation Express. The files generated by the CORE Generator will automatically be copied into the selected project directory.
2. From the CORE Generator “Options” menu, select “Output Format”, and check the following options:



Select either VHDL or Verilog Instantiation template.

3. From the CORE Generator Options menu, select **System Options**, and set the Project Path to point to your Foundation Express project directory.



4. Select the module you want to generate by navigating to the desired module and clicking onit.

You may click the SPEC button on the CORE Generator toolbar to review the module's datasheet. Double-click on the selected module to call up its parameterization window. When you have entered all the parameters required by the module, press the Generate button.

Note: Do not name your Module with a Unified Library Name, as this will cause the Synthesizer to use the Unified Library XNF file instead of the one generated by the CORE Generator.

A VHDL or Verilog snippet (module_name.VHI or module_name.VEI) and a Netlist File (.XNF) will be created and copied into the CORE Generator project directory. The snippet contains the component declaration as well as the Port Map / Module declaration for the module that has been selected. This snippet can be Copied and Pasted into the top-level HDL file as shown in the following examples.

VHDL Example:

Shown below are the VHDL Instantiation Template and a top-level VHDL design that instantiates the XNF file from the CORE Generator.

```
******** 8 Bit Adder VHDL Snippet: adder8.vhi ************

component adder8 port (
      a: IN std_logic_VECTOR(7 downto 0);
      b: IN std_logic_VECTOR(7 downto 0);
      s: OUT std_logic_VECTOR(8 downto 0);
      c: IN std_logic;
      ce: IN std_logic;
      ci: IN std_logic;
      clr: IN std_logic);
end component;

yourInstance : adder8 port map (
      a => a,
      b => b,
      s => s,
      c => c,
      ce => ce,
      ci => ci,
```

```
      clr => clr);

**********************************************

**********  Top Level VHDL file: adder8_top.vhd ******

Library IEEE;
use IEEE.std_logic_1164.all;

entity adder8_top is
        port (ina, inb: in  STD_LOGIC_VECTOR (7 downto 0);
              clk, enable, carry, clear: in STD_LOGIC;
              qout: out STD_LOGIC_VECTOR (8 downto 0));
end adder8_top;

architecture BEHAV of adder8_top is

-- Instantiate the adder8.xnf file.

component adder8 port (
    a:   IN  std_logic_VECTOR(7 downto 0);
    b:   IN  std_logic_VECTOR(7 downto 0);
    s:   OUT std_logic_VECTOR(8 downto 0);
    c:   IN  std_logic;
    ce:  IN  std_logic;
    ci:  IN  std_logic;
    clr: IN  std_logic);
end component;

begin

u1 : adder8 port map (
          a   => ina,
          b   => inb,
          s   => qout,
          c   => clk,
          ce  => enable,
          ci  => carry,
          clr => clear);

end BEHAV;

******************************************************
```

Verilog Example:

Shown below are the Verilog Instantiation Template and a top-level Verilog design that instantiates the XNF file from the CORE Generator. Also shown is a module declaration file for the XNF file. This file, named `module_name.v`, is required to define the port directions of the XNF file. This file can be created by cutting and pasting the module declaration section of the .VEI file into the file called `module_name.v`.

```
******** 8 Bit Adder Verilog Snippet: adder8.vei ************

module adder8 ( a, b, s, c, ce, ci, clr);

input [7:0] a;
input [7:0] b;
output [8:0] s;
input c;
input ce;
input ci;
input clr;
endmodule

// The following is an example of an instantiation :

adder8 YourInstanceName (
      .a(a),
      .b(b),
      .s(s),
      .c(c),
      .ce(ce),
      .ci(ci),
      .clr(clr));

**********************************************

**********  Top Level Verilog file: adder8_top.v ******

module adder8_top(ina, inb, clk, enable, carry, clear, qout);

 input [7:0] ina;
 input [7:0] inb;
 input clk;
 input enable;
 input carry;
 input clear;
```

```
 output [8:0] qout;

// instantiate the adder8.xnf file

adder8 U1 (
      .a(ina),
      .b(inb),
      .s(qout),
      .c(clk),
      .ce(enable),
      .ci(carry),
      .clr(clear));

***************************************************

******** Instantiation Module Declaration: adder8.v ************

module adder8 ( a, b, s, c, ce, ci, clr);
input [7:0] a;
input [7:0] b;
output [8:0] s;
input c;
input ce;
input ci;
input clr;
endmodule

***********************************************
```

Compiling the Design in Foundation Express

1. Create a new project, or open an existing one in Foundation Express.
2. Add all HDL files to be synthesized for the project.

   Note: Do NOT add the XNF files created by the CORE Generator to the Express project. Also, do NOT add any HDL simulation files.
3. Verilog Only: Add a .v module declaration file for each instantiated block.
4. Select the top level entity and select Create Implementation to generate a new implementation.
5. Optimize the implementation.
6. Write out the XNF file for this implementation.

7. The XNF file written by Express and the XNF file(s) created by the CORE Generator are required as inputs to the XACTstep M1 Implementation Tools, and should all be located in the same directory when 1 the design is input to M1.

For additional information on the Foundation Express flow, please refer to the Foundation Express User Guide. For more details on the Alliance FPGA Express flow, please refer to the Quick Start Guide for Xilinx Alliance Series v1.4.

6.4 Synopsys FPGA Compiler VHDL Flow

Example Design Illustrating VHDL Implementation & Simulation Design Flows

This document describes the following processes:

1. How to create a serial distributed arithmetic FIR filter with the Xilinx CORE Generator, and obtain:

   - An implementation netlist (.XNF)

   - A VHDL instantiation 'snippet' (.VHI)

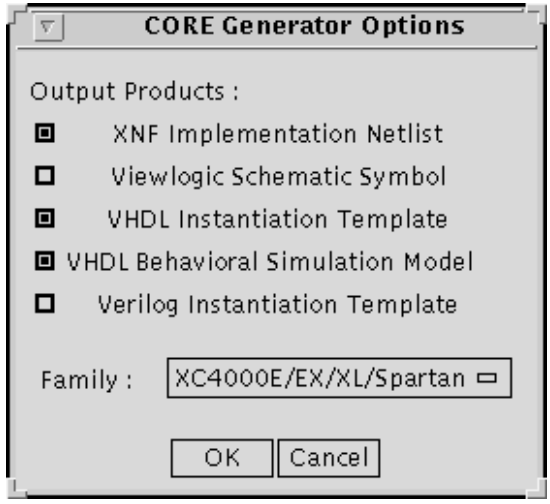
   - A VHDL behavioral model (.VHD)

2. How to embed the filter within a larger VHDL design, and synthesize the design using Synopsys FPGA Compiler.

3. How to simulate the entire design, including the embedded filter, using Synopsys VHDL System Simulator (VSS).

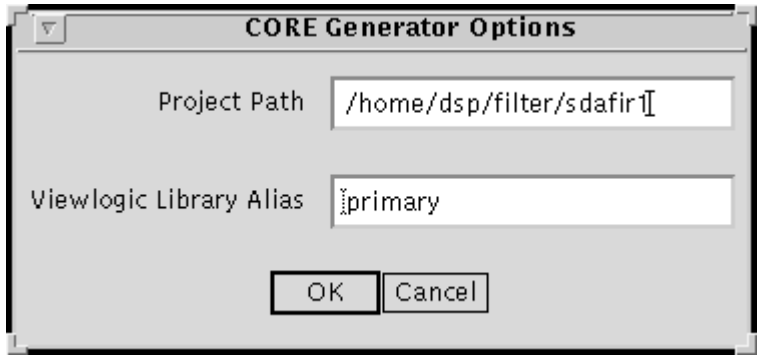
Create an SDA FIR filter with the Xilinx CORE Generator

Invoke the Xilinx CORE Generato. Before you start the process of building the filter, check that the desired output products have been selected from the **Options -> Output_Format...** menu:



These selected options correspond with an XNF implementation netlist, a VHDL instantiation snippet, and a VHDL behavioral model respectively.

Also you need to set the Project Path to point to the directory you wish, to hold the files generated by CORE Generator. The recommended location for this directory is the directory that will hold your Synopsys design. The Project Path can be set from the **Options -> System Options** menu in the CORE Generator.



To create the filte, navigate through the tree of available modules to the **SDA FIR Filter Single-Channel** leaf.

The path you should follow through the tree is:

```
Core Generator Library -> LogiCORE -> DSP -> Filters -> FIR Filters -
> Serial Distributed Arithmetic
```

Before you can build the filter, you must describe the details of the filter's construction and present this information to the CORE Generator. This can be achieved by creating a core-definition file (.COE) containing the required information. An example .COE file for an SDA FIR Filter is shown below.

```
-- EXAMPLE.COE ------------------------------------
Component_name = example;
Number_of_taps = 26;
Input_Width = 12;
Output_Width = 12;
Coef_Width = 12;
Symmetry = false;
Radix = 16;
coefdata = 346,EDA,0D6,F91,079,FC8,053,FE2,
       03C,FF2,02D,FFB,022,002,01A,005,
```

```
       014,008,00F,008,00B,008,009,006,
       008,FFE;
-------------------------------------------------------------------------------
```

This .COE fil instructs the CORE Generator to build a 26-tap, non-symmetric filter, which takes 12-bit input data, has 26 12-bit coefficients, and which generates a full precision output truncated to 12-bits. The coefficients are shown in hexadecimal format, and are in increasing time order.

Double-click on the **SDA FIR Filter Single-Channel** entry in the CORE Generator module browser. This calls up the SDA FIR Filter parameterization window. Click on the **Load Coefficients** button and specify the `EXAMPLE.COE` file shown above. Inspect the various fields in the window to ensure that they have all taken on the values specified in the .COE file. Finally click on **Generate** to begin building the filter.

When the CORE Generator has finished, you should find the following files in your project directory :

`example.xnf` - The implementation netlist

`example.vhi` - The VHDL instantiation snippet

`example.vhd` - The VHDL behavioral model

Instantiate the filter within a larger VHDL design and synthesize with Synopsys

A simple example of a larger design containing the filter that was created above is shown below. The sections of code peceded by comments were cut-and-pasted from the VHDL instantiation snippet file EXAMPLE.VHI. In the second commented section, the default signal names used in the snippet have been edited to match the signal names used in the actual design.

```
-- DESIGN.VHD ------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY design IS
  PORT( data_in      : IN  std_logic_vector(11 DOWNTO 0);
        data_out     : OUT std_logic_vector(11 DOWNTO 0);
        new_data     : IN  std_logic;
```

```
     rdy_for_data : OUT std_logic;
    result_ready : OUT std_logic;
        control      : IN  std_logic;
        ck           : IN  std_logic );
END design;

ARCHITECTURE filter OF design IS

  SIGNAL filtered_data : std_logic_vector(11 DOWNTO 0);
  SIGNAL nc_1 : std_logic;  -- Just a dangling net
  SIGNAL nc_2 : std_logic;  -- Just a dangling net



  -- ------------------------------------------------ --
  -- Component declaration provided by CORE Generator   --
  -- Filename: EXAMPLE.VHI                              --
  -- ------------------------------------------------ --

  component example port (
        data: IN std_logic_VECTOR(11 downto 0);
        result: OUT std_logic_VECTOR(11 downto 0);
        nd: IN std_logic;
        rfd: OUT std_logic;
        sinf: IN std_logic;
        sinr: IN std_logic;
        soutf: OUT std_logic;
        soutr: OUT std_logic;
        ck: IN std_logic;
        rdy: OUT std_logic);
  end component;

BEGIN

  data_out <= filtered_data WHEN control='1' ELSE data_in;
```

```
  -- ------------------------------------------------ --
  -- Component instantiation provided by CORE Generator --
  -- Filename: EXAMPLE.VHI ------------------------------

  u0 : example PORT MAP (
        data => data_in,
        result => filtered_data,
        nd => new_data,
        rfd => rdy_for_data,
        sinf => nc_1,
        sinr => nc_2,
        soutf => OPEN,
        soutr => OPEN,
        ck => ck,
        rdy => result_ready );

END filter;
-------------------------------------------------------------
```

Assuming that the Xilinx/Synopsys Interface product has been installed correctly, and that a `.synopsys_dc.setup` file exists in your project directory that points to the target synthesis library for the desired device architecture, you can use the following dc_shell script to synthesize the above design:

```
-- DESIGN.SCR --------------------------------
read -f vhdl design.vhd
set_dont_touch u0
set_port_is_pad all_inputs() + all_outputs()
set_pad_type -clock ck
insert_pads
compile
replace_fpga
write -h -f xnf -output design.sxnf
quit
-----------------------------------------------------
```

Synopsys may issue a warning about the lack of a design called *example* in its database. This warning may be ignored since *example* is the filter that was created earlier, and which will be merged into the design once synthesis is complete.

When FPGA Compiler completes, it should leave behind a file called `design.sxnf` which is the Xilinx Netlist Format result of the synthesis process. To process this netlist, merging-in the filter created earlier in the process, run the following command:

(XILINX M1)

```
ngdbuild -p <target_part_type/speed_grade> design.sxnf
```

Simulate the Entire Design, Including the Embedded Filter

Simulation of a design written in VHDL, and which contains a Xilinx Core Module (such as the one created in step 1), with a behavioral simulator can be achieved by using behavioral models that are created by the module generator. The VHDL behavioral model mimics the behavior of the specific module whose parameters were entered via the module generator's parameterization window. If any details of an existing module are subsequently altered via the module generator's parameterization window, ensure that the **VHDL Behavioral Simulation Model** option is checked on the

CORE Generator Options->Output format window

since the behavioral model must be recreated to reflect these changes.

Using Synopsys' VHDL System Simulator (VSS), you are required to declare the whereabouts of the WORK library, usually a subdirectory in your project directory. The location is specified in a file called `.synopsys_vss.setup` found in the project directory. Any other libraries containing, for example, behavioral models for instantiated primitives, should also be declared here. Note that some behavioral models created by the module generator need access to a library of supporting functions called 'XUL'. The whereabouts of this library will need to be recorded in the `.synopsys_vss.setup` file.

An example `.synopsys_vss.setup` file is shown below:

```
-- .SYNOPSYS_VSS.SETUP --------------------------

TIMEBASE        = NS
TIME_RES_FACTOR = 0.1
WORK    > DEFAULT
DEFAULT : ./WORK
XUL     : ../XUL_VSS

-------------------------------------------------
```

In preparation for simulating a design containing a Xilinx CORE Generator behavioral model, you need to analyze the library of supporting functions 'XUL'. To do this, select/create a directory in which to store the analyzed version of this library, and record it in the `.synopsys_vss.setup` file, as shown above. (The location of this directory is not too important, although it should ideally be a central location where other designers can access it.) To analyze the 'XUL' library, in a directory containing a `.synopsys_vss.setup` file which contains the correct 'XUL' pointer, type the following command:

```
vhdlan -w XUL <COREGEN_Install_Path>/ip/xilinx/xul/ul_utils.vhd
```

To simulate the design shown in step 2, which contains the Xilinx Core Module created in step 1, you should first analyze the behavioral model for the filter, then analyze the synthesizable code itself.

Note: It is important that the simulation model be used for simulation *ONLY*. Behavioral models created by the Module Generator are inadequate for synthesis purposes.

Finally, the testbench - the simulation vectors written in sequential VHDL - should be analyzed. A testbench suitable for exercising the design shown in step 2 is included below, followed by the sequence of commands necessary to simulate the design with Synopsys VSS:

```
-- TESTBNCH.VHD ---------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY testbench IS
END testbench;
```

```
ARCHITECTURE for_example OF testbench IS

  COMPONENT design
    PORT( data_in : IN  std_logic_vector(11 DOWNTO 0);
         data_out : OUT std_logic_vector(11 DOWNTO 0);
         new_data : IN  std_logic;
     rdy_for_data : OUT std_logic;
     result_ready : OUT std_logic;
          control : IN  std_logic;
     ck : IN  std_logic );
  END COMPONENT;

  SIGNAL data_in      : std_logic_vector(11 DOWNTO 0);
  SIGNAL data_out     : std_logic_vector(11 DOWNTO 0);
  SIGNAL new_data     : std_logic;
  SIGNAL control      : std_logic;
  SIGNAL rdy_for_data : std_logic;
  SIGNAL result_ready : std_logic;
  SIGNAL ck           : std_logic;

  CONSTANT half_clock_period : TIME := 100 NS;

BEGIN

  uut : design PORT MAP ( data_in => data_in,
                          data_out => data_out,
                          new_data => new_data,
                      rdy_for_data => rdy_for_data,
                      result_ready => result_ready,
                           control => control,
                                ck => ck );
```

```
stimulus : PROCESS
BEGIN

  data_in  <= "000000000000";
  new_data <= '0';
  control  <= '1';
  ck       <= '1';

  WAIT FOR half_clock_period;

  -- ------------------------ --
  -- Fill the filter with 0's --
  -- ------------------------ --

  FOR i IN 0 TO 26 LOOP
    WHILE rdy_for_data = '0' LOOP
      ck <= NOT ck; WAIT FOR half_clock_period;
      ck <= NOT ck; WAIT FOR half_clock_period;
    END LOOP;
    new_data <= '1';
    ck <= NOT ck; WAIT FOR half_clock_period;
    ck <= NOT ck; WAIT FOR half_clock_period;
    new_data <= '0';
  END LOOP;

  -- ------------- --
  -- Do an impulse --
  -- ------------- --

  WHILE rdy_for_data = '0' LOOP
    ck <= NOT ck; WAIT FOR half_clock_period;
    ck <= NOT ck; WAIT FOR half_clock_period;
  END LOOP;
```

```
    data_in  <= "011111111111";
    new_data <= '1';

    ck <= NOT ck; WAIT FOR half_clock_period;
    ck <= NOT ck; WAIT FOR half_clock_period;

    data_in  <= "000000000000";
    new_data <= '0';

    -- ---------------------------- --
    -- Continue with a field of 0's --
    -- ---------------------------- --

    FOR i IN 0 TO 26 LOOP
      WHILE rdy_for_data = '0' LOOP
        ck <= NOT ck; WAIT FOR half_clock_period;
        ck <= NOT ck; WAIT FOR half_clock_period;
      END LOOP;
      new_data <= '1';
      ck <= NOT ck; WAIT FOR half_clock_period;
      ck <= NOT ck; WAIT FOR half_clock_period;
      new_data <= '0';
    END LOOP;

    WAIT; -- The End

  END PROCESS; -- stimulus

END for_example;

CONFIGURATION cfg_testbench OF testbench IS
  FOR for_example
```

```
  END FOR;
END cfg_testbench;

-------------------------------------------------------
```

Using VSS, the commands required to analyze and simulate this design are:

```
vhdlan example.vhd
vhdlan design.vhd
vhdlan testbnch.vhd
vhdldbx cfg_testbench
```

At the VHDL debugger (vhdldbx) command line, issue the following commands to begin the simulation and observe the resulting waveforms:

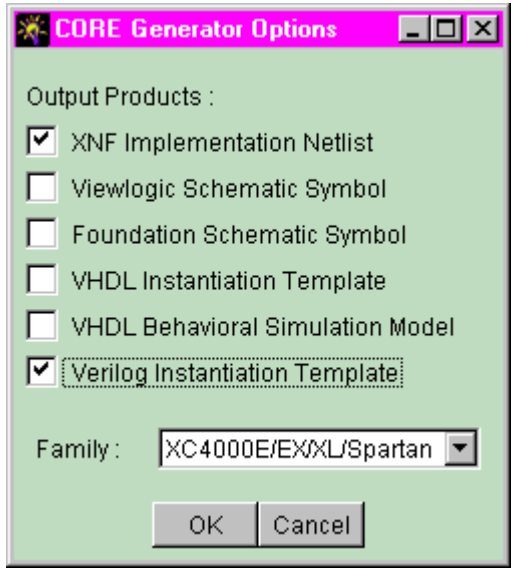
```
trace *'signal
run
```

6.5 Synopsys FPGA Compiler Flow (Verilog)

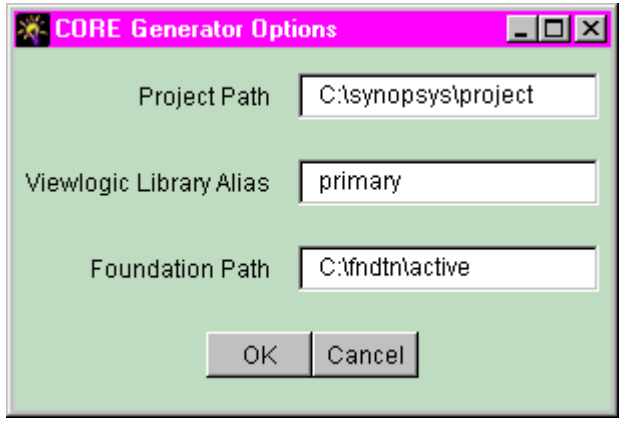
Generate the Desired Core Module

Invoke the Xilinx Core Generator. Before you start the process of building a module, select the following output formats from the Options -> Output_Format... menu:

- XNF Implementation Netlist
- Verilog Instantiation Template



You will also need to set the Project Path to point to a directory where the files generated by the CORE Generator should be written. The recommended location for this directory is the directory that will hold your Synopsys design. The Project Path can be set from the Options -> System Options menu in the CORE Generator:



Select the module you wish to generate by navigating to it using the CORE Generator module browser and clicking on the desired module.

You may click the SPEC button on the CORE Generator toolbar to review the module's datasheet. Double-click on the selected module to display its parameterization window. When you have entered all the parameters required by the module, press the **Generate** button.

Note: Do not name the Module with a Unified Library Name; if you do, the Synthesizer will use the XNF file corresponding to the Unified Library component, instead of the XNF file created by the CORE Generator.

A Verilog snippet (module_name.VEI), and a Netlist File (.XNF) are created and copied into the user-specified CORE Generator project directory.

The Verilog snippet contains the module declaration with accompanying port definitions, as well as sample instantiation syntax for the module:

```
******** 8 Bit Adder Verilog Snippet: ad8.vei ************

module ad8 ( a, b, s, c,      ce, ci, clr);
<-Verilog module declaration for the CORE

input [7:0] a;
input [7:0] b;
output [8:0] s;
input c;
```

```
input ce;
input ci;
input clr;
endmodule


// The following is an example of an instantiation :

ad8    YourInstanceName (
      .a(a),
      .b(b),
      .s(s),
      .c(c),
      .ce(ce),
      .ci(ci),
      .clr(clr));

**********************************************
```

The Verilog module declaration for the CORE Generator core function should be placed in a separate .v file named *module_name.v*.

Instantiate the module within your design

The instantiation template in the .VEI file can be cut and pasted into your Top Level verilog design as shown in the following example:

```
**********************************************
**********  Top Level Verilog file: adder8_top.v ******

module adder8_top(ina, inb, clk, enable, carry, clear, qout);

 input [7:0] ina;
 input [7:0] inb;
 input clk;
 input enable;
 input carry;
 input clear;
 output [8:0] qout;
```

```
// instantiate the adder8.xnf file

adder8 u0 (
      .a(ina),
      .b(inb),
      .s(qout),
      .c(clk),
      .ce(enable),
      .ci(carry),
      .clr(clear));

endmodule  /* adder8_top */

*****************************************************

******** Instantiation Module Declaration: adder8.v ************

module adder8 ( a, b, s, c, ce, ci, clr);
input [7:0] a;
input [7:0] b;
output [8:0] s;
input c;
input ce;
input ci;
input clr;
endmodule
```

Synthesize the Design

When compiling the design, read in the design starting with the modules at the bottom of your hierarchy (bottom to top). Be sure to attach a "dont_touch" property to all CORE Generator modules to prevent these from being re-optimized by Synopsys.

After the compile step in Synopsys, do a remove_design on the ad8 design before writing out the .sxnf file. The remove_design step prevents Synopsys from writing out an empty sXNF file for the CORE Generator mod-

ule(s). (This step is required for Verilog designs only.)

You can use the following dc_shell script to synthesize the design:

```
-- DESIGN.SCR ----------------------------------------------------:

read -f verilog add8_top.v
set_dont_touch u0
set_port_is_pad all_inputs() + all_outputs()
set_pad_type -clock clk
insert_pads
compile
replace_fpga
remove_design ad8
write -f xnf -h -o add8_top.sxnf
```

Post-Synthesis Functional Simulation

To simulate the design, please follow these steps:

- Run NGDBUILD on the synthesized design netlist (.SXNF):

  ngdbuild -p 4028expg299-2 add8_top.sxnf

In this example the target part is an XC4028EX part in a PG299 package and -2 speed grade

- Run NGD2VER on the .NGD file produced by NGDBUILD.

NOTE: Use the -tf option so that ngd2ver will generate a test fixture template for you.

NOTE: If you are using Verilog-XL, it is recommended that you also specify the -ul option so that the path the simulation libraries used will be written to your .v file.

ngd2ver -tf -ul add8_top.ngd add8_topf

- Edit the test fixture template file and add your test vectors for the design.
- Perform gate level functional simulation

From this point onward, the flow is the same as for designs containing only Unified Library components.