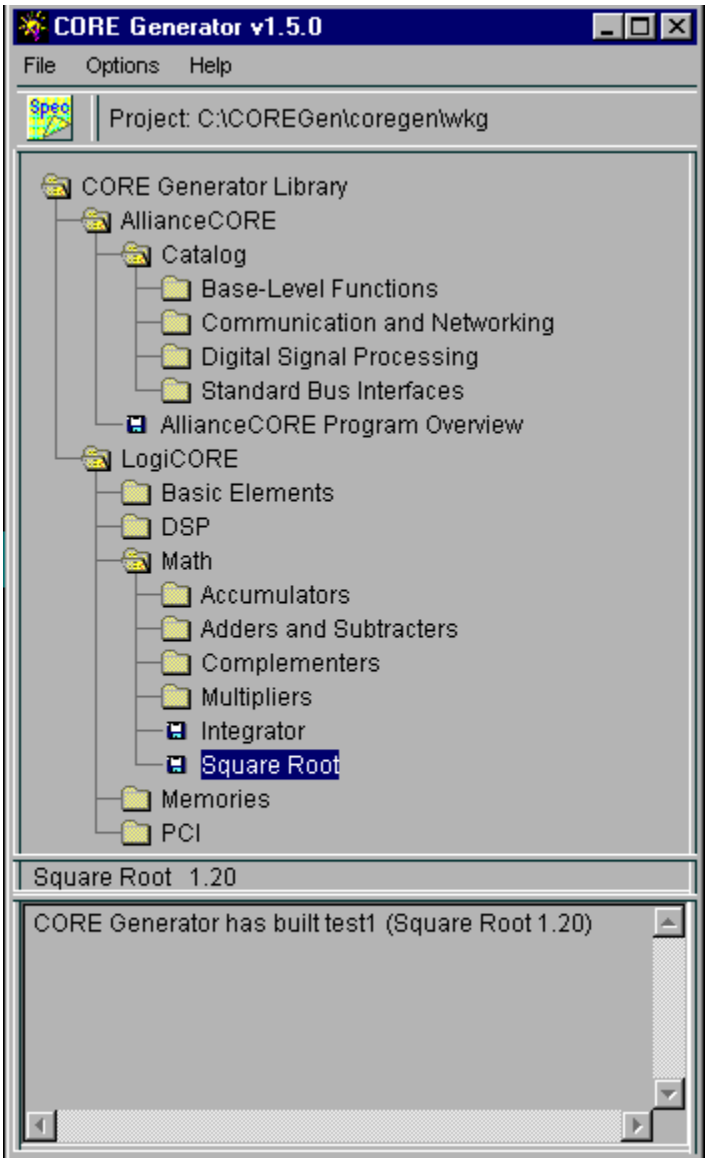
CORE Generator™ System User Guide V1.5.2i

Table of Contents

1 Introduction

1.1 Overview

Welcome to version 1.5 of the Xilinx CORE Generator ™ System, an easy to use design tool that delivers parameterizable cores optimized for Xilinx FPGAs. The application's main graphical user interface (GUI) allows central access to cores, spec sheets, important options and help functions.



The Xilinx CORE Generator System (CGS) provides the designer with a catalog of ready-made functions ranging in complexity from simple arithmetic operators such as adders, accumulators and multipliers, to system-level building blocks including filters, transforms and memories.

Note: The words **function** and **core** are used interchangeably in this guide to mean a design entity like a multiplier or FIR filter which the CORE Generator System can generate for the designer.

Cores are organized by type into folders that expand or contract on demand. Detailed information on each core is contained in a specification or data sheet which can be accessed by clicking on the **Spec** icon in the toolbar or by selecting **Help -> Spec Sheet** from the menu bar. This launches the Adobe Acrobat Reader and calls up the data sheet for that core. Datasheets include:

- Functional information
- Area and performance data
- Pinouts and interface signal names
- Details of how to use the core in an application

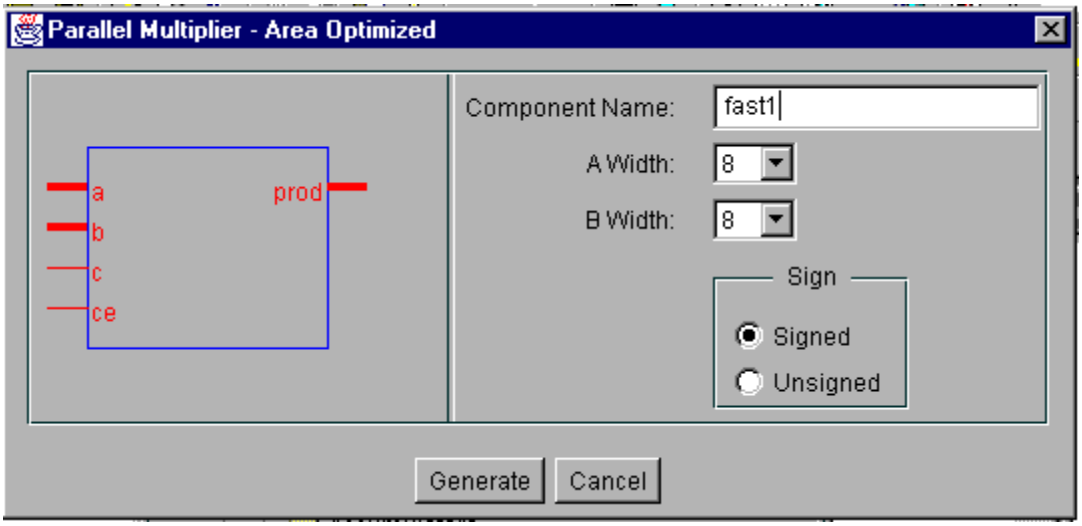
One of the unique features of the CORE Generator System is its ability to tailor a generic functional building block such as a FIR filter or a multiplier to meet the needs of your application and to simultaneously deliver the highest possible levels of performance and area efficiency. This is accomplished by the use of Xilinx's core-friendly FPGA architectures and by the application of Xilinx Smart-IP™ technology. Smart-IP™ technology leverages:

- Xilinx FPGA architectural advantages such as look-up tables (LUTs), distributed RAM, segmented routing and floorplanning information
- Relative location constraints and expert logic mapping to guarantee performance of a given core instance in a given Xilinx FPGA.

Smart-IP™ technology provides benefits including:

- Physical layout optimized for high performance
- Predictability
- Efficient power utilization through compact design and use of less interconnect
- Device size independent performance
- Ability to use multiple cores without deterioration of performance
- Reduced compile time over competing architectures.

Parameterization provides the ability to generate cores which meet design flexibility needs and which meet design size constraints.



For each core the CORE Generator System delivers:

- Verilog or VHDL behavioral simulation models
- Foundation or Viewlogic schematic symbols
- A customized EDIF netlist.
- VHDL or Verilog instantiation templates

This allows the CORE Generator System to fit easily into your preferred design environment.

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 simpler and faster than ever before.

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 System from this location.

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

Xilinx encourages you to check the CORE Generator System WEB page before starting a new design to verify that you have the latest version of each 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 System WEB page.

2 System Requirements

2.1 All Platforms

Adobe Acrobat ver 3.0.0 or later (Adobe Acrobat 3.0.1 included on CORE Generator System 1.5 CD in the *vendor/acrobat* directory)

2.2 Requirements for PCs

- PC with Windows 95 or Windows NT v4.0.0 or later
- IBM-compatible Pentium-class machine recommended, 486 PC acceptable

Table 2. 1. Memory Requirements for PCs

Xilinx Device	RAM	Virtual Memory
XC4003E/L through XC4008E/L XC4005XL through XC4008XL	32 MB	32 MB - 64 MB
XC4010E/L through XC4025E/L XC4028EX through XC4036EX XC4010XL through XC4028XL XC4085XL XC4013XLA through XC4028XLA Spartan/XL	64 MB	64 MB - 128 MB
XC4036XL through XC4062XL XC4036XLA through XC4085XLA	128 MB	128 MB - 256 MB
XC40110XV through XC40150XV XCV50, XCV100, XCV150, XCV200, XCV300	128 MB	256 MB
XC40200XV through XC40250XV XCV400, XCV600	256 MB	400 MB
XCV800, XCV1000	512 MB	800 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 cores, as well as for concurrent operation of other applications (for example, MS Word or Excel).

2.3 Requirements for Workstations

- Solaris 2.5 or 2.6
- HP-UX 10.2
- Ultra Sparc (or equivalent)
- HP715 (or equivalent)

Table 2.2 Memory Requirements for Workstations

Xilinx Device	RAM	Swap Space
XC4000E/L XC4028EX through XC4036EX XC4005XL through XC4028XL XC4013XLA through XC4028XLASpartan/XL	64 MB	64 MB - 128 MB
XC4036XL through XC4062XL XC4036XLA through XC4062XLA XCV50 through XCV300	128 MB	128 MB - 256 MB
XC4085XL/XLA XC40110XV through XC40150XV XCV400 through XCV600	256 MB	400 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 cores, as well as for concurrent operation of other applications such as Netscape Communicator.

2.4 CAE Platform Requirements

The following CAE platform versions are supported:

Platform	Version
• Foundation (schematic)	F1.5 (Required)
• Foundation simulation	F1.5 (Required)
• Foundation Express	F1.5
• Synopsys FPGA Express (Verilog,VHDL)	2.1.1 or later
• Synopsys FPGA Compiler (Verilog,VHDL)	1997.01 or later
• Workview Office	7.4 or later
• Viewlogic Powerview	6.0 or later

3 Installation Instructions

3.1 Xilinx CORE Generator System Requirements for PCs

Refer to the Section 2.2, *Requirements for PC's*, for information on operating system and memory requirements for the CORE Generator System.

3.2 Xilinx CORE Generator System Installation on PCs

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

1. Insert the CORE Generator CD-ROM into the CD-ROM drive.
2. Run **Setup.exe** (located in the CD-ROM root directory) unless install starts automatically.

3.3 Xilinx CORE Generator System Requirements for Workstations

Refer to Section 2, *Requirements for Workstations,* for information on the operating system and memory requirements for the CORE Generator System.

3.4 Xilinx CORE Generator System Installation on Workstations

All workstation installations must be done while logged in with “root” authority. Installation of the Xilinx CORE Generator System software is completed in two steps.

1. Mount the CORE Generator CD-ROM.
2. Run **install** (located in the CD-ROM root directory) by typing "./install".

For detailed information on how to mount and unmount a CD-ROM, and how to run the various installation programs, see the installation section of the M1.5 Release Document.

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 System.

Please bookmark this location in your web browser**:**

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

Check this web page regularly for updates.

Note: We also encourage you to check the CORE Generator System WEB page before beginning a new design to ensure you have the latest versions of the cores, core data sheets, and 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 System WEB page.

4 Project Management

The CORE Generator System determines its operational settings by reading the coregen.ini file. This file is created during the installation of the CORE Generator System. There are two ways to change the CORE Generator System settings.

1. Modify the parameters in the **Options->System Options…** menu item at the top of the core browser window. To save these settings permanently in a coregen_<user_name>.ini file, click on the “Save Settings upon OK” checkbox.
2. Manually modify the parameters by editing the coregen.ini file found in the install_dir/coregen/wkg directory.

All files created by the CORE Generator System on a PC are stored by default in:

<CORE_Generator_Install_Path>/coregen/wkg

On workstations, the default location for files created by the CORE Generator System is the directory in which you started up the CORE Generator software.

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

All files created by the CORE Generator System on a workstation are stored by default in the directory from which you are running the CORE Generator System.

4.1 The COREGEN.INI File

The CORE Generator System uses several components from Xilinx and other third party packages, including Acrobat, Viewlogic and Alliance Series 1.5 tools for Viewlogic users. (These should be installed before installing the CORE Generator System. During CORE Generator System installation, the locations of these components are recorded in the coregen.ini file.

The coregen.ini file is written to the following directory:

<CORE_Generator_Install_Path>/coregen/wkg

The following parameters are determined during installation, and usually do not need to be changed: CoreGenPath, FoundationPath, AcrobatPath, and AcrobatName. Other settings, such as SelectedProducts, ProjectPath, BusFormat, and XilinxFamily will usually need to be changed by the user.

Changes to your output options and system options are best done through the CORE Generator **Output format** and **System Option** forms. You can access these forms by selecting **Options->Output format…** to get the Output format form, and **Options-> System Options…** for the System Options form. Click on the "Save Settings upon OK" checkbox to make these settings permanent.

When you click on “Save Settings upon OK”, the CORE Generator System will try to determine the current “User” name. It will then will create a user-specific

coregen.ini file under the file name of **coregen_<user_name>.ini** in the CORE Generator System coregen\wkg directory (PC's), or the current working directory (UNIX workstations). If the “User” cannot be determined, a coregen.ini file with the name **coregen_nouser.ini** will be created. You can view the coregen.ini file to determine what the default values of the various settings are at installation.

You may also edit the coregen.ini file directly using a text editor to make changes if needed, but such an approach tends to be more error-prone.

Coregen.ini Syntax Summary

```
SET CoreGenPath = <some platform-specific path to the
$COREGEN\coregen directory>

SET FoundationPath = <some user-specific path to the Foundation
tools>

SET ProjectPath = <some user-specific path to the current project>

SET TargetSymbolLibrary = <Viewlogic library name>

SET AcrobatPath = <some user-specific path to Acrobat install>

SET AcrobatName = <Acrobat executable name>

SET SelectedProducts = [ImpNetlist | FoundationSym| VHDLSym |
VHDLSim |ViewSym | VerilogSim | VerilogSym]

SET XilinxFamily = [All_XC4000_Families |All_Spartan_Families
|All_Virtex_Families]

SET BusFormat = [BusFormatAngleBracket | BusFormatSquareBracket
|BusFormatParen | BusFormatNoBracket]

SET OverwriteFiles =[true | false]
```

Sample coregen.ini file:

```
SET FoundationPath = C:\FNDTN\ACTIVE

SET ProjectPath = C:\cg95_cur\coregen\wkg

SET TargetSymbolLibrary = primary

SET XilinxFamily = All_XC4000_Families


SET AcrobatPath = C:\Acrobat3\Reader\

SET AcrobatName = AcroRd32.exe

SET SelectedProducts = ImpNetlist FoundationSym
```

CoreGenPath - This setting specifies the path to the "coregen" subdirectory, which is located one level below the root directory where the CORE Generator System software is installed. This value is set during installation, and should not be changed.

Example: `SET CoreGenPath = C:\software\cgen_c15\coregen`
Options: `Path to the installed location of the CORE Generator software`

ProjectPath - This setting defines the project or working directory. All output files produced by the CORE Generator System are placed here. If you change the Project Path parameter to a new location, all new CORE Generator output files from that point on will be deposited into the new CORE Generator project directory. The default value of Project Path that is set during the installation of the CORE Generator System for a PC is:

<CORE_Generator_Install_Path>/coregen/wkg

On workstations the default value of ProjectPath is **"."**

Look here for output files created by the CORE Generator System when ProjectPath is set to its default value.

Example: `SET ProjectPath = C:\projects\dsp\filterv1_0\wkg`
Options: `Path to working directory`

XilinxFamily - This setting defines the target Xilinx part type Family selected.

Example: `SET XilinxFamily = All_Spartan_Families`

GUI Options: `All XC4000 Families, All Spartan Families, All Virtex Families`

Coregen.ini Options: `All_XC4000_Families, All_Spartan_Families, All_Virtex_Families`

BusFormat - This setting defines the Bus bracket format used in the EDIF implementation netlist. This option is set automatically when any of the following output formats are selected: Viewlogic Symbol (**ViewSym**), Foundation Symbol (**FoundationSym**), VHDL instantiation template (**VHDLSym**), or Verilog instantiation template (**VerilogSym**).

Example: `SET BusFormat = BusFormatAngleBracket`

Coregen.ini Options: `BusFormatAngleBracket,`
`BusFormatSquareBracket,`
`BusFormatParen,`
`BusFormatNoBracket`

SelectedProducts - This setting defines the type of output files to be created each time a core is built. Multiple output formats should be specified on the same line, separated by a blank space. This parameter is case sensitive.

Example: `SET SelectedProducts =ImpNetlist FoundationSym`

Options: ImpNetlist, FoundationSym, VHDLSym, VHDLSim, ViewSym, VerilogSim, and/or VerilogSym

ImpNetlist: Implementation Netlist. This is the gate level netlist that will be used to implement the logic of the particular core that the COREGenerator System has created. In an HDL synthesis design flow, an HDL instantiation of the core references this netlist as a “black box”; in a schematic design flow, a schematic symbol references this netlist.

ViewSym: Viewlogic Schematic Symbol. When specified, this option will create a Viewlogic schematic symbol that can be used in your Viewlogic schematic capture tools to instantiate the core netlist.

FoundationSym: Foundation Schematic Symbol. When specified, this option will create a Foundation schematic symbol that can be used in your Foundation schematic capture tools to instantiate the core netlist.

VHDLSym: VHDL Instantiation Template. When specified, this option will create a VHDL instantiation template that can be used to instantiate the core netlist in a 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 functionality of the core netlist.

VerilogSym: Verilog Instantiation Template. When specified, this option will create a Verilog instantiation template that can be used to instantiate the core netlist in a Verilog design.

VerilogSim: Verilog Behavioral Simulation Model. When specified, this option will create a Verilog behavioral simulation model, which can be used to verify the functionality of the core netlist.

ViewlogicLibraryAlias - This setting defines the name of the Viewlogic library alias that will be attached to all Viewlogic symbols and WIR files created by the CORE Generator, and thus the Viewlogic user library to which these files will be stored.There should be a matching alias in the viewdraw.ini file for your current project. The default value is “primary”.

Example: `SET ViewlogicLibraryAlias=primary`
Options: `Any valid 8-character, alphanumeric string.`

FoundationPath - This setting defines the path location of the Foundation CAE tools. It must be specified in order 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 location of the Acrobat tools. It must be specified for the core data sheets and online help documentation to load properly.

```
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:    Name of the Acrobat executable.
```

Advanced: Multiple COREGEN. INI Files

When the CORE Generator System is launched, it looks for a **coregen_<user_name>.ini** or **coregen.ini** file using the following search order, using the first one it finds:

PC's:

File Name	Location
coregen_user.ini	startup directory specified by Windows shortcut
coregen.ini	startup directory specified by Windows shortcut
coregen_user.ini	coregen\wkg directory
coregen.ini	coregen\wkg

UNIX Workstations:

File Name	Location
coregen_user.ini	current working directory
coregen.ini	current working directory
coregen_user.ini	coregen/wkg
coregen.ini	coregen/wkg

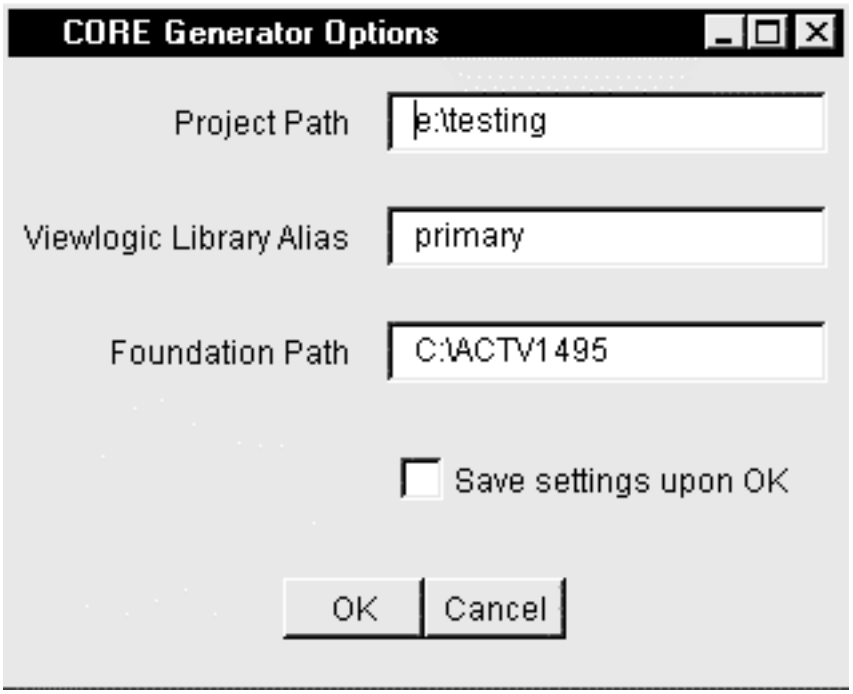
On Windows 95 and NT platforms, the directory from which the coregen.ini or coregen_<user_name>.ini file is read is defined in the "Start in" setting in the Windows desktop shortcut, or in the Windows Start menu shortcut for the CORE Generator System.

You can change the startup directory for CORE Generator System either in the CORE Generator shortcut on your Windows desktop, or in the CORE Generator entry in the Windows TaskBar by editing the "Start in" property. (It is usually not necessary to do this.)

Note: The working directory specified in the shortcut must be set to a directory containing either a **coregen.ini** file or a **coregen_< user _name>.ini** file. If the working directory specified does not contain a valid copy of at least one of these .ini files, the CORE Generator System will not be able to start up.

4.2 Set the CORE Generator System Options

System options for a CORE Generator System session are set by selecting **Options->System_Options** from the top of the core browser window. Be aware that the options set through this option menu are only applicable to the current CORE Generator session. To set these options permanently for all subsequent sessions, select “Save Settings upon OK” in the **Options->System Options** form.



In this form you will find the following options:

ProjectPath - This setting defines the project or working directory. All output files produced by the CORE Generator System are placed here. If you change the Project Path parameter to a new location, all new CORE Generator output files from that point on will be deposited into the new CORE Generator project directory. The default value of Project Path that is set during the installation of the CORE Generator System for a PC is:

On workstations the default value of Project Path is: **"."**

Look here initially for output files created by the CORE Generator System when ProjectPath is set to its default value.

```
Example:    SET ProjectPath = C:\projects\dsp\filter\v1_0\wkg
Options:    Path to working directory.
```

TargetSymbolLibrary- - This setting defines the name of the Viewlogic library alias that will be attached to all Viewlogic symbols and WIR files created by the CORE Generator System , and thus the Viewlogic user library to which these files will be stored There should be a matching alias in the viewdraw. ini file for your current project. The default value is “primary”.

```
Example:    SET V TargetSymbolLibrary =primary
Options:    Any valid 8-character, alphanumeric string.
```

FoundationPath - This setting defines the 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.
```

To make your changes permanent, select “**Save settings upon OK**” in the **Options->System_Options** menu.

4.3 Set the CORE Generator Output Options

The CORE Generator System can create several different types of output. The desired outputs for a particular project may be selected from the **Options-> Output_Format** dialog window. To make any changes permanent, select “Save Settings upon OK” in the **Options->System_Options** menu dialog window.

The default selections are EDIF Implementation Netlist and VHDL Instantiation Template. The following is a description of each file format:

Edif Implementation Netlist - This is the gate level netlist that will be used to implement the logic for the particular core that the CORE Generator System has created. The HDL templates or schematic symbols will point to this netlist. The EDIF output should normally be selected but is not required. There may be a case where you only generate a Verilog or VHDL behavioral model.

```
Output:     <CoreName>.edn
```

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 core netlist.

Note: Viewlogic simulation is not supported for Virtex cores.

```
Output:     wir <CoreName>.1
            sym <CoreName>.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 core netlist as well as a simulation file for functional simulation.

```
Output:     <CoreName>.alr
            libproject_name.sym
```

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

Output: `<CoreName>.vhi`

VHDL Behavioral Simulation Model - When selected this option will create a VHDL simulation model, which can be used to verify the functional behavior of the core netlist. This file is optimized for simulation only, and should not be used for synthesis. Trying to synthesize this behavioral model will give you suboptimal results.

Output: `<CoreName>.vhd`

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

Output: `<CoreName>.vei`

Verilog Behavioral Simulation Model - When selected this option will create a Verilog simulation model which can be used to verify the functional behavior of the core netlist. This file is optimized for simulation only, and should not be used for synthesis. Trying to synthesize this behavioral model will give you suboptimal results.

Output: `<CoreName>.v`

The Family drop-down box allows you to set the CORE Generator's core browser to show only those cores that may be targeted to the selected family of devices. The supported families of devices are:

- All XC4000 Families
- All Spartan Families
- All Virtex Families

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

To make these options permanent, check on the "**Save settings upon OK**" checkbox in the **Options->Output Formats** menu.

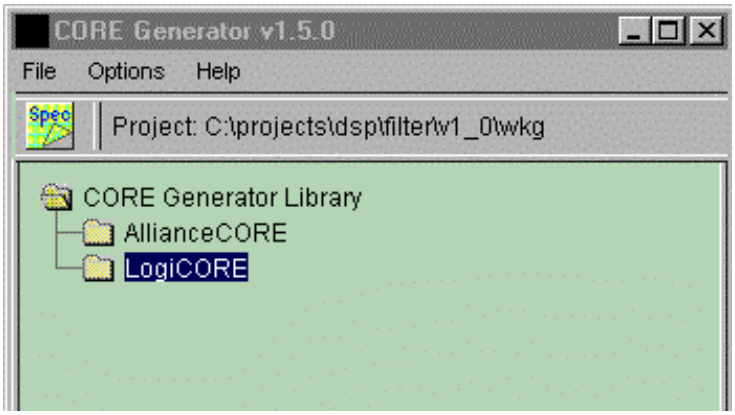
5 Using the CORE Generator System

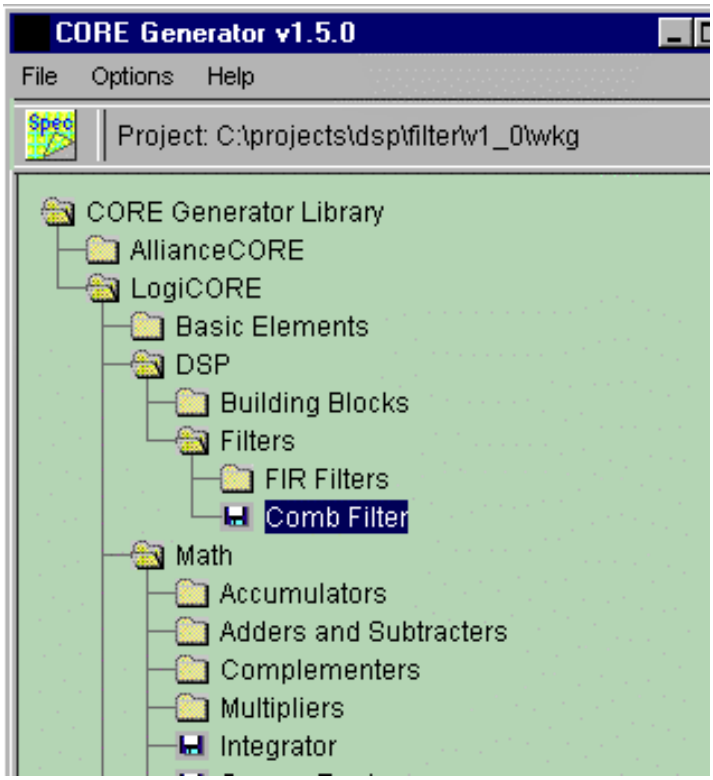
5.1 Core Browser Tree

The most common view of the CORE Generator System is the **core browser** window. This window allows you to browse the many cores that are available from the CORE Generator installation. Cores that fall into particular application categories are grouped into folders to assist you in locating the core appropriate to your needs.

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

Note: The folder may be closed by double clicking again on the open folder icon.

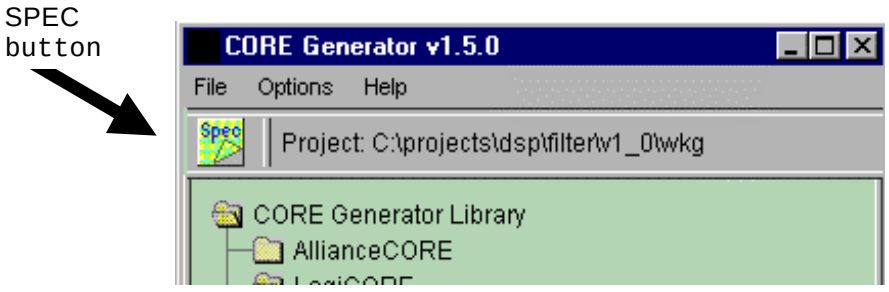




Finally, when the desired core has been located it can be selected by single clicking on its icon.

5.2 Accessing Core Data Sheets

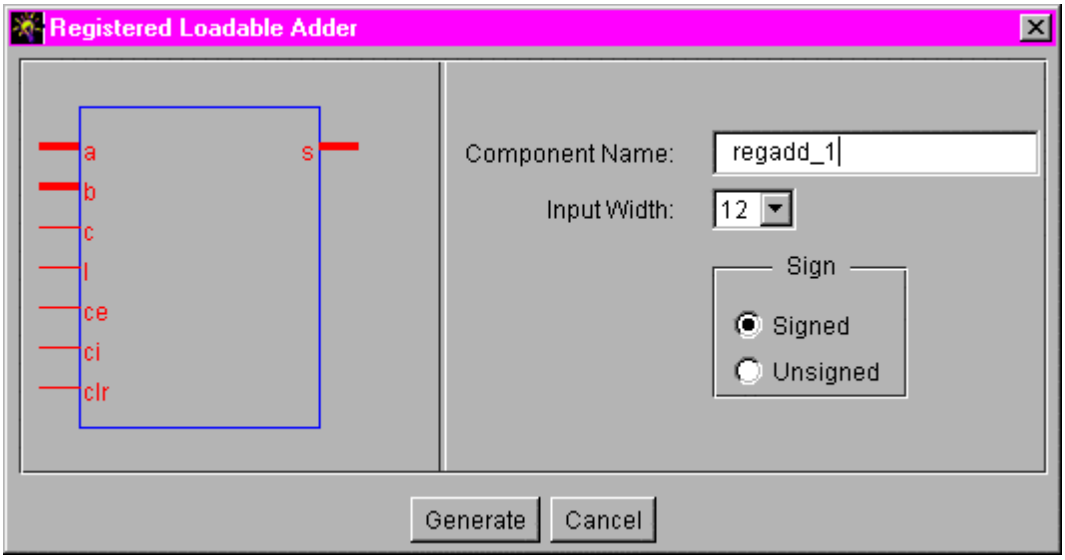
To view a data sheet for a selected core first close any core GUIs that are open, then select the core in the core browser, then click on the SPEC button on the CORE Generator toolbar. Alternatively, with the core selected in the core browser, you can also click on the **Help ->Spec Sheet…**. This action will launch the Acrobat Reader application and display the core's data sheet.



5.3 Parameterizing a Core

Most cores have a parameterization window. The parameterization window shown below was launched by navigating to the CORE Generator's 'Registered Loadable Adder' entry. Double-clicking on the core's icon or descriptive text will reveal the

parameterization window for that core. While the parameterization windows will be unique for each core, there are some characteristics that are common to all.



For example, the **Component Name field** allows you to assign a name to the core that you create. Files that the CORE Generator System creates for a particular core 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 an alpha character: a-z (No Capital letters)
- May include (after the first character): 0- 9, _(underscore)

5.3.1 Illegal or Invalid Data Values

All parameterization windows flag illegal or invalid data in the same fashion. The affected field is 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 System to create files of the requested types. Pressing **Cancel** will return you to the core browser window without generating any files.

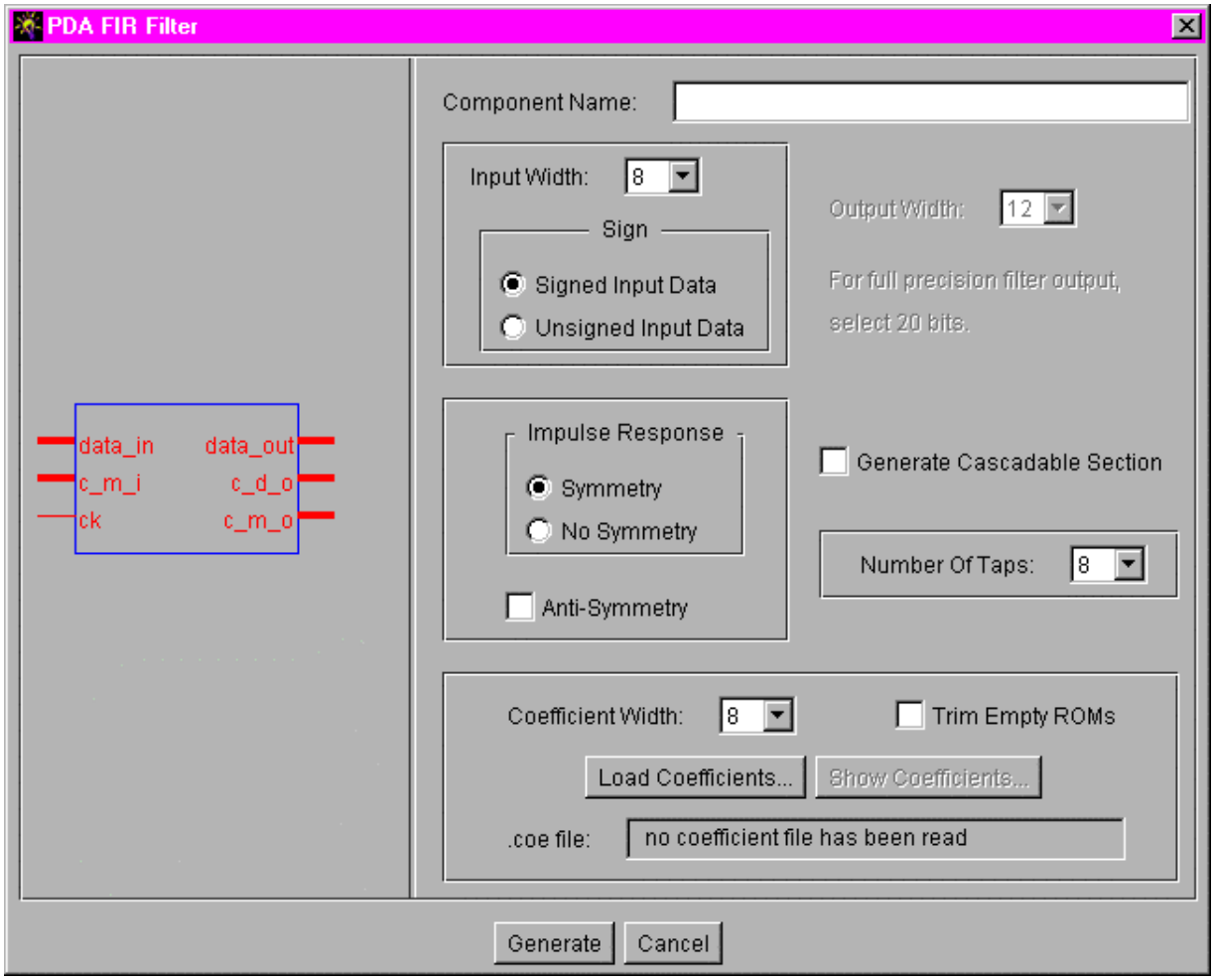
For information about a specific core's parameterization window, such as upper and lower limits for certain fields, see the core's datasheet.

5.4 COE Files

Some cores require a significant amount of information in order to completely, after behaviors, specify their behavior. Cores of this type (PDA FIR, SDA FIR, XC4000 RAM and ROM, and Virtex Block RAM, for example) usually require multiple coefficients or initialization values. To specify the values more conveniently, you can load a .COE file using the **Load Coefficients…** button on the parameterization window.

Additional information about a particular core's .COE file can be found in that core's datasheet. For examples of PDA FIR, SDA FIR, RAM, and ROM .COE files, as well as Virtex block RAM COE files, look in the `coregen\wkg` directory.

A FIR Filter parameterization window is shown below.



The parameterization window for a FIR filter has a **Load Coefficients** button. Files containing coefficient information should be ASCII text files and have a .COE extension. The format for the .COE file for a FIR filter is illustrated below:

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

Data for a memory module is specified with a MemData keyword.

Note: CoefData or MemData keywords must appear at the end of the file as any keywords that follow them will be ignored. Any text after a semicolon is treated as a comment and will be ignored.

An example .COE file that might be used to parameterize a PDA 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;
Signed_Coefficient =true
coefdata= 1,-3,7,9,78,80,127,-128;
```

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

```
********** EXAMPLE: SDA FIR **********

component_name=sdafir;
number_of_taps=6;
radix=10;
input_width=10;
output_width=24;
coef_width=11;
symmetry =false;
coefdata=-1,18,122,418,-40,3;
```

A sample .COE file that might be used to parameterize an XC4000 or Spartan 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;
```

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

```
********** EXAMPLE: Virtex Block Ram **********

component_name=blkram;
Depth =256;
Data_Width =32;
Radix =16;
Default_Data =FFF;
MEMORY_INITIALIZATION_VECTOR =FF0,F0F,0FF,FF4,F4F,4FF,FF8,F8F,8FF;
```

5.5 <core_name>.log

The CORE Generator System will produce a <Core_Name>.log for every core that is generated. This log file is a record of all the options used to create the core. This file is useful when you want to verify all the options that were used when the core was generated. Comment lines begin with “#”. Any output format or system options lines start with the “SET” keyword. These options will match the options set in the coregen_<User_Name>.ini. The lines that start with “GSET” are the options that are passed from the core parameterization dialog window. Any data read in from a .COE file are also listed here.

```
********** EXAMPLE: .LOG file **********

# CORE Generator v1.5.0

# Username = Mars

SET ProjectPath = C:\COREGen\coregen\wkg

SET XilinxFamily = All_XC4000_Families

SET BusFormat = BusFormatAngleBracket

SET SelectedProducts = ImpNetlist VHDLSym

SET TargetSymbolLibrary = primary

GSET ModuleTaxonomy = CORE Generator Library/LogiCORE/DSP/Filters/FIR
Filters/Parallel Distributed Arithmetic/PDA FIR Filter
```

```
# CORE Generator Library/LogiCORE/DSP/Filters/FIR Filters/Parallel
Distributed Arithmetic/PDA FIR Filter 1.28

GSET Coef_Width = 12

GSET CoefData9 = 0

GSET CoefData8 = 0

GSET CoefData7 = -30

GSET CoefData6 = 83

GSET CoefData5 = -56

GSET CoefData4 = 121

GSET CoefData3 = -111

GSET CoefData2 = 214

GSET CoefData1 = -294

GSET CoefData0 = 838

GSET Signed_Input_Data = true

GSET Number_Of_Taps = 16

GSET Antisymmetry = false

GSET Trim_Empty_ROMs = false

GSET Cascade = false

GSET Symmetry = true

GSET Output_Width = 20

GSET Input_Width = 8

GSET Component_Name = videofft
```

6 Design Flows

6.1 Viewlogic Schematic Flow

The Viewlogic interface outputs are generated by a script, **vllink**, which calls both Viewlogic tools and Xilinx M1 implementation programs.

As a result, on Windows platforms you must have both the Viewlogic and Xilinx M1 software installed on your machine to generate the outputs required to integrate a core into a Viewlogic design. On UNIX platforms, your environment must be set up to run the Viewlogic tools and the Xilinx M1 software as well as the CORE Generator System. If any of these tools are either not installed or not set up in your environment, diagnostic errors may be reported in the **vllink.log** file written to your project directory.

For more information on Viewlogic and Xilinx M1 setup, please refer to the *Alliance Quick Start User Guide.*

1. Create a directory for your Viewlogic project.

   Eg: `c:\wvoffice\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 set up through the Project Manager GUI as described below:

   a. Open a New project

   b. Add a configured FPGA library from the list. If the xc4000XL is selected, then the xc4000x, logiblox, simprims, builtin and xbuiltin libraries are added automatically.

   c. Add the project directory as a writeable library and give it the alias "primary". Move it to the top of the library search order.

   d. Save the project in the Viewlogic Project Manager.

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

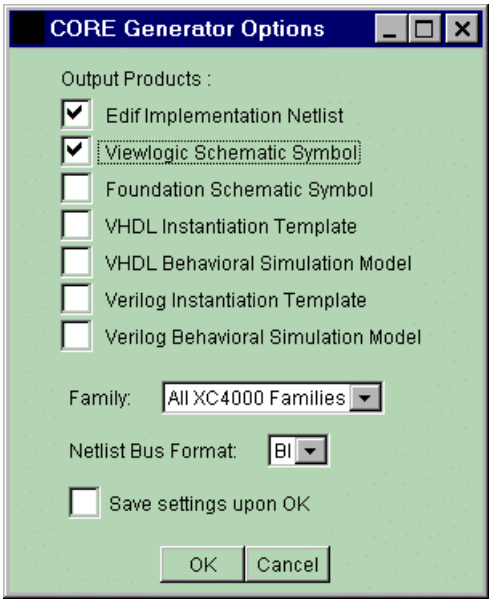
```
dir [p] c:\wvoffice\project (primary)
dir [rm] %XILINX%\viewlog\data\xc4000x (xc4000x)
dir [r] %XILINX%\viewlog\data\logiblox (logiblox)
dir [rm] %XILINX%\viewlog\data\simprims (simprims)
dir [rm] %XILINX%\viewlog\data\builtin (builtin)
dir [rm] %XILINX%\viewlog\data\xbuiltin (xbuiltin)
```

Note: The (primary) alias is very important since the CORE Generator System will look for it in order to define in which directory the symbol and simulation files will be

copied. This alias should match the one specified in the CORE Generator System Options under Viewlogic Library alias.

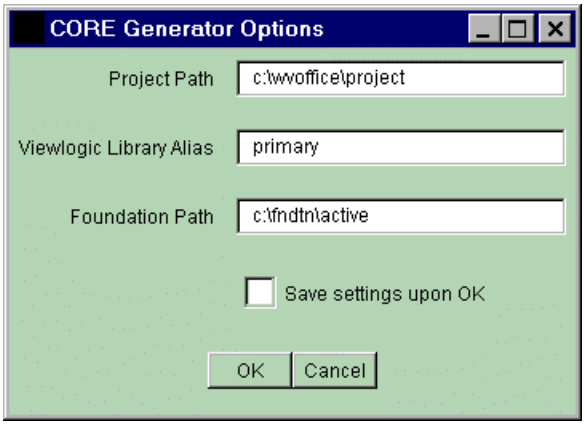
3. Set the Output Format.

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



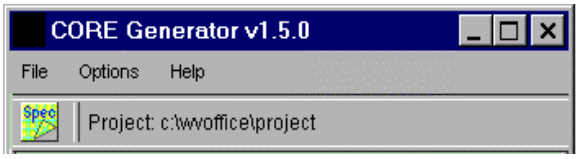
4. Set the Project Path and the Viewlogic Library Alias

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



5. Select the module you want to generate by navigating around the 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 reveal its parameterization window.



   When you have entered all the parameterization details required by the module, click the **Generate** button.

6. Output Files

   A Viewlogic Symbol, a Viewlogic simulation file and a Netlist File (.EDN) will be created.

   - The symbol of type composite is created and placed in the SYM subdirectory within the Viewlogic project. This symbol will not necessarily match the CORE Generator module symbol shown in the datasheets in shape and pin order. It is possible to manually modify your symbol using the Viewlogic Symbol Editor as needed.

   - The simulation file is created from the EDN file and placed in the WIR subdirectory within the Viewlogic project.

   - The Netlist (.EDN) used for implementation, is placed directly in the Viewlogic project directory.

   Note 1: The WIR file is used by Viewlogic to perform Functional simulation and should not be deleted. In order to generate this file the CORE Generator System needs to access some M1 executables and may fail if this tool is not set up properly.

   Note 2: Workview Office 7.4 and Powerview 6.0 do not support Virtex simulation, so only a symbol and a netlist are generated for these cores. You must use either VHDL or Verilog to simulate Virtex designs in Viewlogic.

   Check the V**llink.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 now on the flow for processing this design is the same as if you were using macros from the Unified Library. Refer to the *Viewlogic Tutorial Guide* for further information.

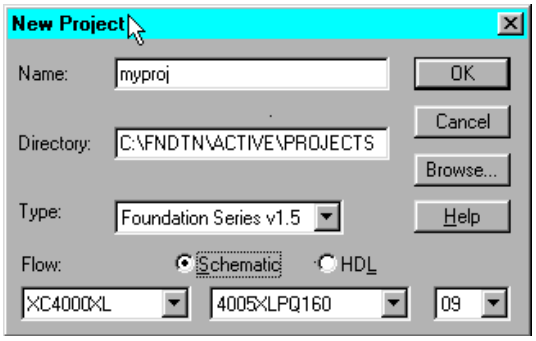
Note: When executing the Viewlogic “Check” program, some error messages will be displayed for every CoreGen Module but can be safely ignored.

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

6.2 Foundation Schematic Flow

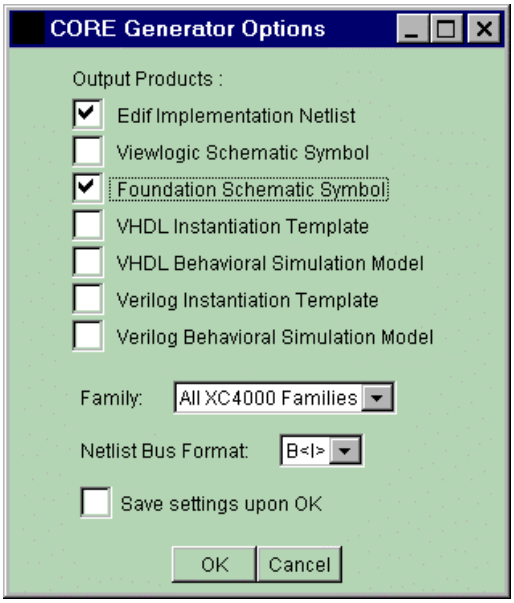
1. Set a Foundation Project

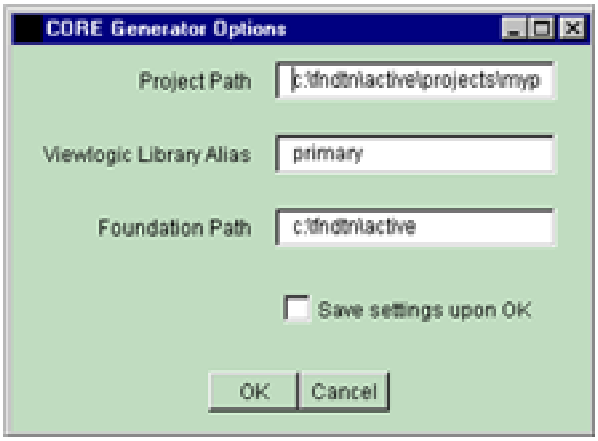
   Create a new project or Select an existing project from the Foundation Project Manager.



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. The selected Project Path should be a valid Foundation project or an error will occur when generating the core.

   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 click the **Generate** button.

5. Output Files

   A Foundation symbol, a Netlist File (.EDN) and a simulation file (.ALR) are created. The symbol is automatically copied to the Foundation Project directory.

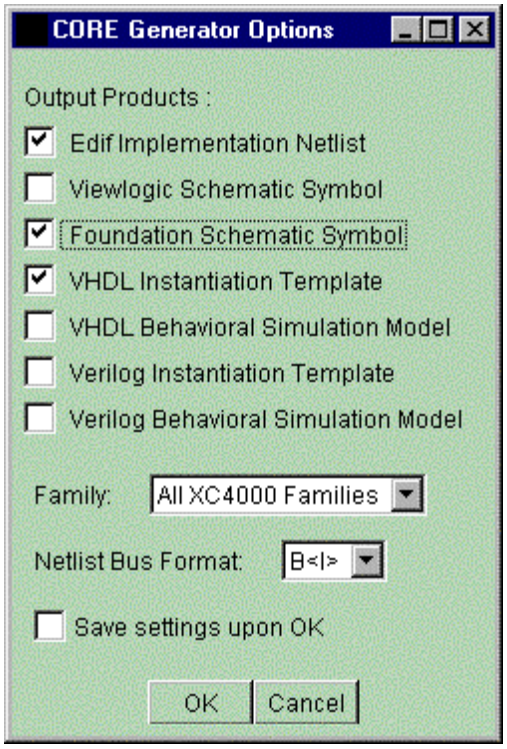
6. Load the Symbol in the Schematic Editor

   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 symbol list for the selected project.

   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 flow for a design containing only Unified Library components.

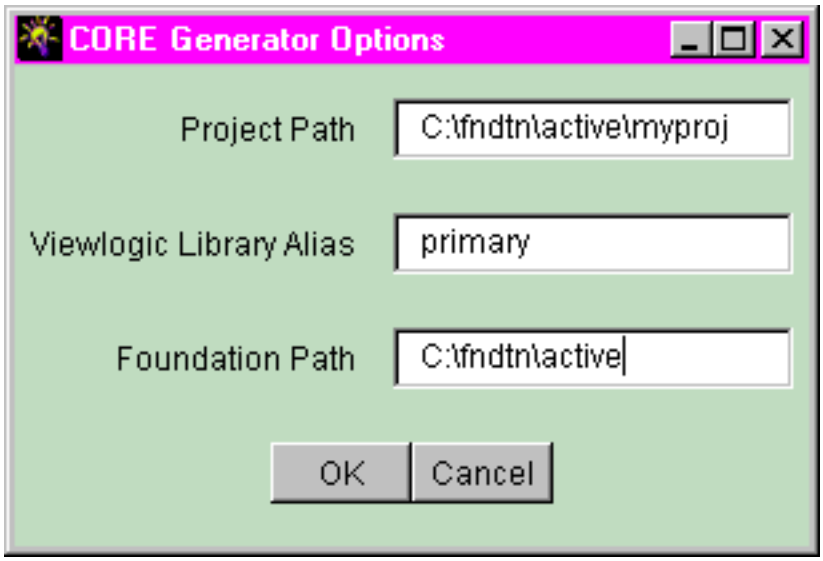
6.3 Foundation Express Flow

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



Note: In order to be able to do a functional simulation of a Coregen module with the Foundation schematic BEFORE running NGBUILD, you **must** also select ‘Foundation Schematic Symbol’ when specifying your desired output formats. This is true even if you are instantiating the Coregen module in a pure VHDL or Verilog design and do not need the schematic symbol. The reason for this is the ALR file, required by the Foundations simulator for the design, is created by NET2SYM and NET2SYM is run only by the Core Generator when you elect to generate a Foundation Schematic Symbol.

3. From the CORE Generator Options menu select **System Options** and set the Project Path to point to your Foundation Express project directory. Enable the “Save settings on OK” checkbox to save these settings for subsequent sessions if desired.



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

   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 click the Generate button.

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

   A VHDL or Verilog instantiation template (module_name.VHI or module_name.VEI) and a Netlist File (.EDN) will be created and copied into the CORE Generator project directory. The instantiation template contains the component declaration as well as the Port Map/Module declaration for the module that has been selected. This instantiation template 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 for a sample COREGEN Adder module, and a top-level VHDL design that instantiates the corresponding EDIF file generated by the CORE Generator System for this module.

```
** 8 Bit Adder VHDL Instantiation template: 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.edn 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 for a sample COREGEN Adder module and the top-level Verilog design that instantiates the EDIF file generated by the CORE Generator System for this module. In the same template file there is a template for a separate module declaration file. This file, named `module_name.v`, is required to define the port directions for the CORE Generator module EDIF file.

The module declaration file can be created by cutting and pasting the module declaration section of the .vei file into a file called `module_name.v`.

```
** 8 Bit Adder Verilog Instantiation template: 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 EDIF files created by the CORE Generator System 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 EDIF file for this implementation.
7. The EDIF file written by Express and the EDIF file(s) created by the CORE Generator System are required as inputs to the XACTstep M1 Implementation Tools, and should all be located in the same directory when the design is input to M1.

For additional information on the Foundation Express flow, refer to the *Foundation Express User Guide*. For more details on the Alliance FPGA Express flow, refer to the *Quick Start Guide for Xilinx Alliance Series v1.5*.

6.4 Synopsys FPGA Compiler VHDL Flow

Example Design Illustrating VHDL Implementation and Simulation Design Flows

This document describes the following processes:

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

   - An implementation netlist (.EDN)

   - A VHDL instantiation 'instantiation template' (.VHI)

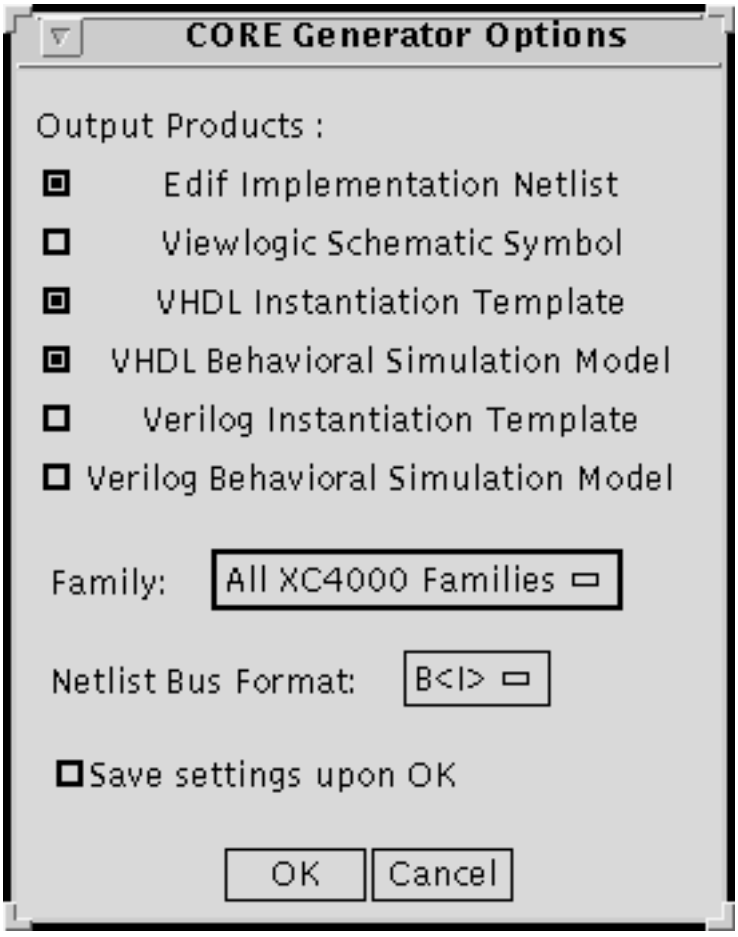
   - A VHDL behavioral model (.VHD)

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

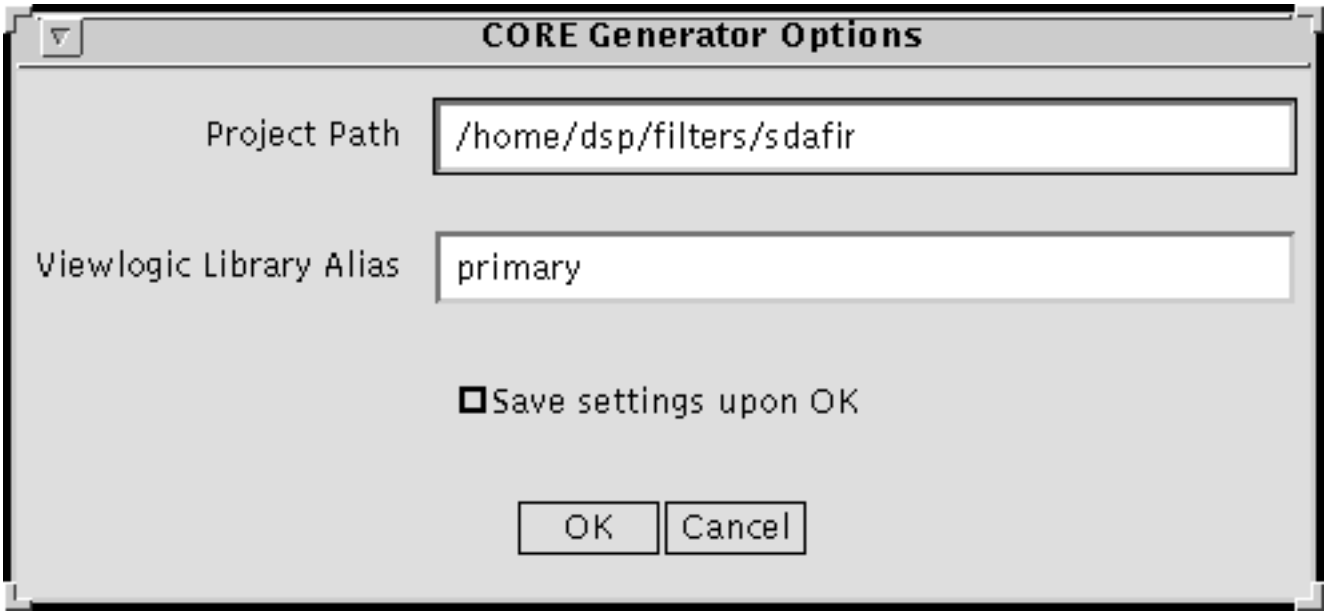
3. How to simulate the entire design, including the embedded filter, using Synopsys VHDL System Simulator (VSS).

Step 1: Create an SDA FIR filter with the Xilinx CORE Generator System

Invoke the Xilinx CORE Generator System . 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 EDN implementation netlist, a VHDL instantiation template, 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 System . 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 System .



To create the filter, 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 System . 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 file instructs the CORE Generator System 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 SDA.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 System has finished, you should find the following files in your project directory :

example.edn - The implementation netlist
example.vhi - The VHDL instantiation template
example.vhd - The VHDL behavioral model

Step 2: 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 preceded by comments were cut-and-pasted from the VHDL instantiation template file EXAMPLE.VHI. In the second commented section, the default signal names used in the instantiation template 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 fpga_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 EDN -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 will 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
```

Step 3: Simulate the Entire Design, Including the Embedded Filter

Simulation of a design written in VHDL, 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 as 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: 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 important, although it should ideally be a central location where other designers can access it.) To analyze the 'XUL' library I(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
```

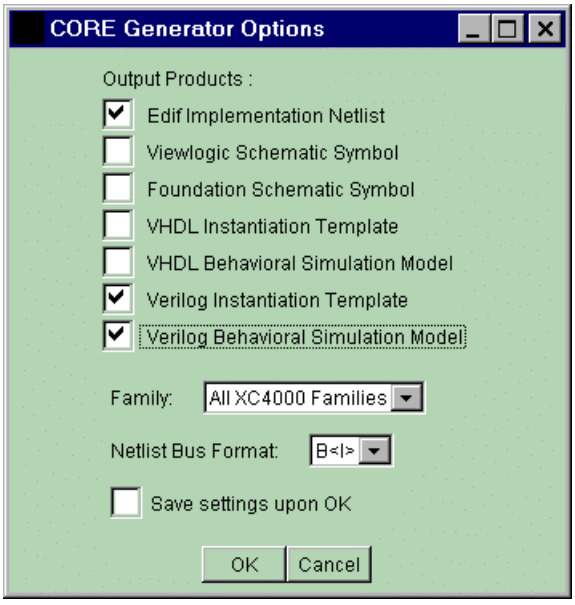
From here on the flow is the same as for designs containing only the Unified Library Components.

6.5 Synopsys FPGA Compiler Flow (Verilog)

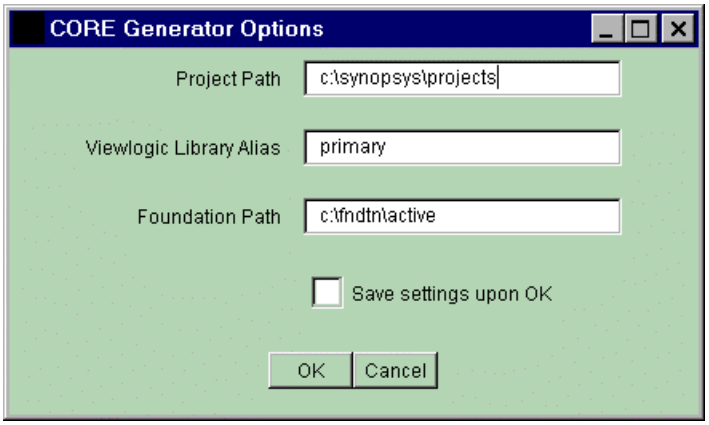
Generate the Desired Core Module

Invoke the Xilinx CORE Generator System . Before you start the process of building a module, select the following output formats from the **Options ->Output Format...** menu:

- EDN Implementation Netlist
- Verilog Instantiation Template
- Verilog Behavioral Simulation Model



You will also need to set the Project Path to point to a directory where the files generated by the CORE Generator System 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 System :



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 click the **Generate** button.

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

A Verilog instantiation template (module_name.vei), and a Netlist File (.edn) are created and copied into the user-specified CORE Generator project directory.

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

```
******** 8 Bit Adder Verilog Instantiation template: 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.EDN 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 module(s). (This step is required for Verilog designs only.)

You can use the following fpga_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
```

The following script can be used target a Virtex part. Additional examples can be found in the $XILINX/synopsys/examples.

```
read -f vhdl design.vhd
set_dont_touch u0
set_port_is_pad all_inputs() + all_outputs()
set_pad_type -clock ck
insert_pads
edifout_design_name = design <- this line allows physical backannotated simulation
write -h -f EDN -output design.sedif
quit
```

Post-Synthesis Functional Simulation

To simulate the design, follow these steps:

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

```
ngdbuild -p <target_part_type_with_speed grade> 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 used by the simulation libraries 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.